Formal Modeling and Security Analysis for Intra-level Privilege Separation

Yinggang Guo, Zicheng Wang, Bingnan Zhong, Qingkai Zeng

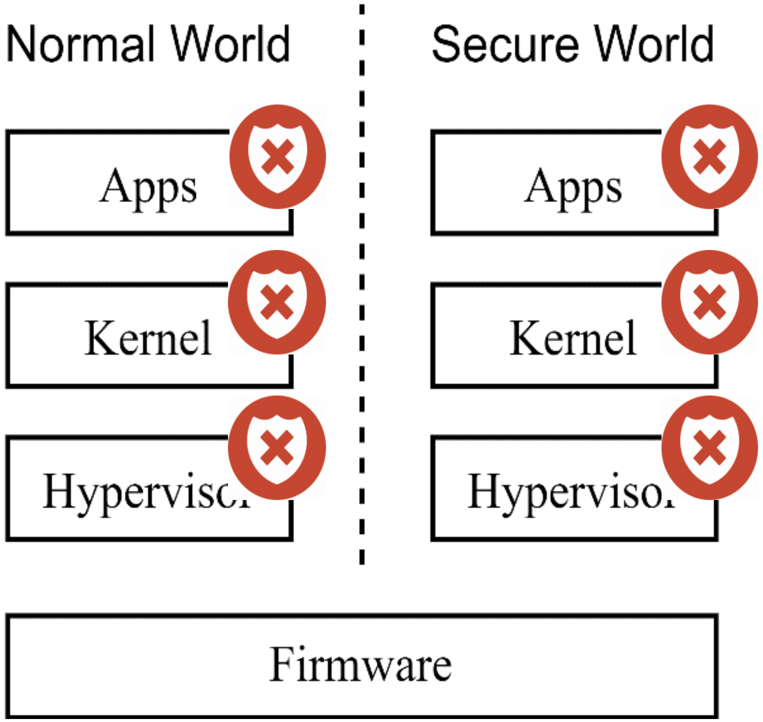
State Key Laboratory for Novel Software Technology

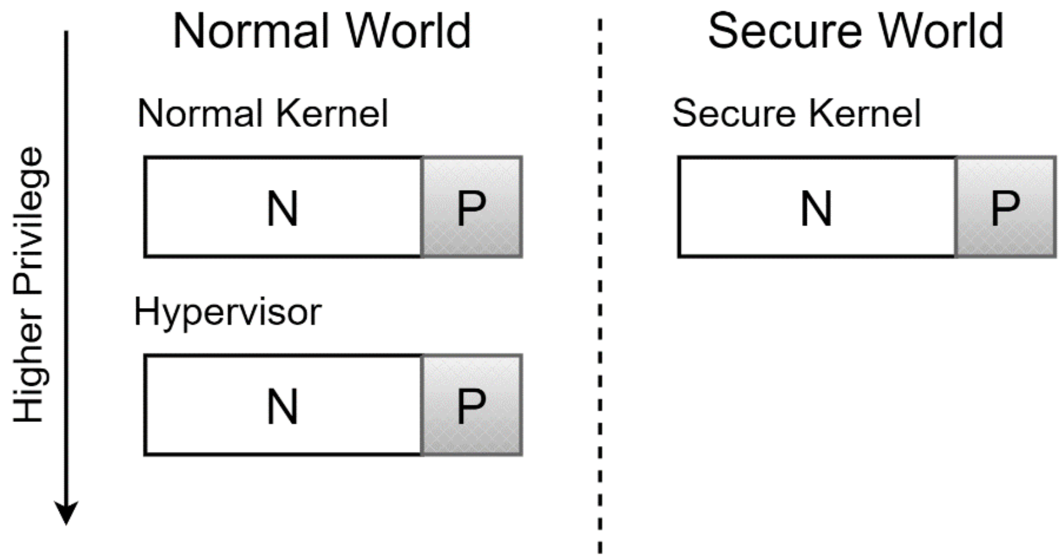
南京大學 NANJING UNIVERSITY

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:

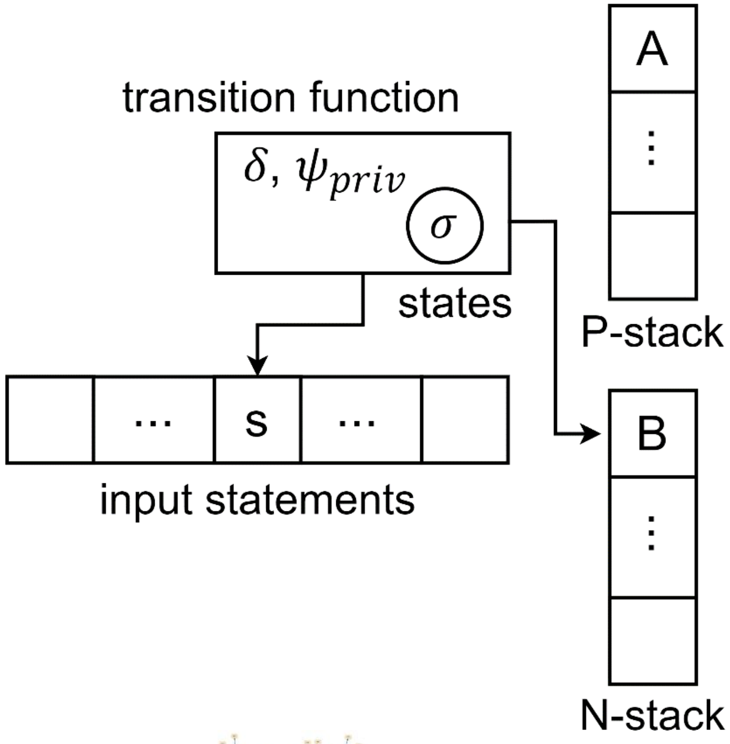
3. ***For comparison***

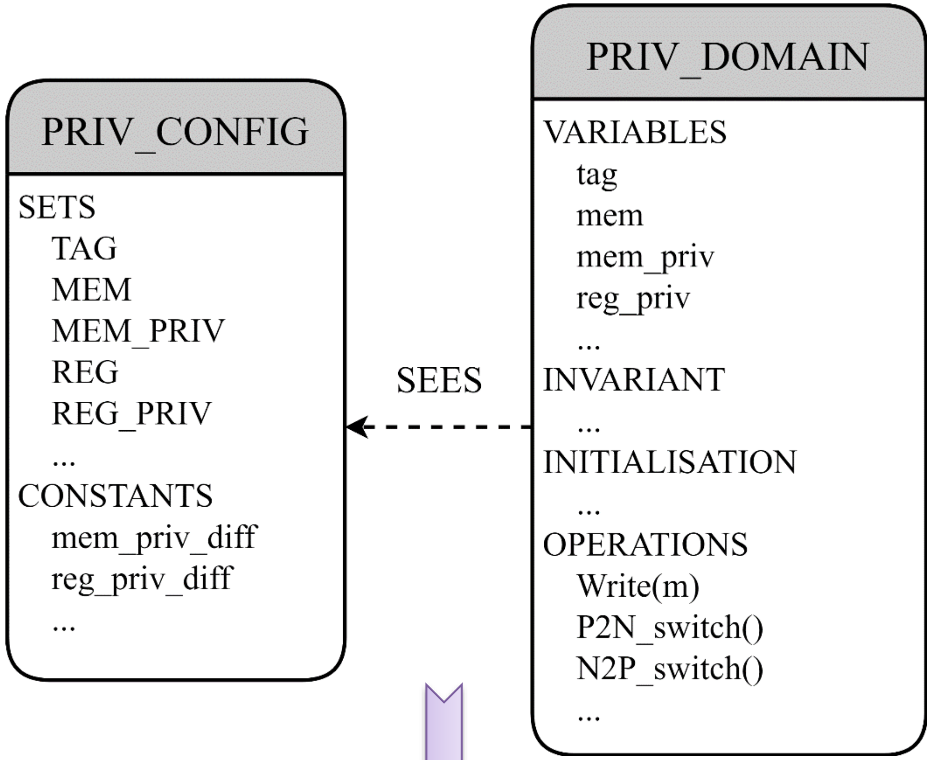
- ➢ difficult to compare the security of different solutions
- ➢ difficult to evaluate potential improvements
    - ✓ A general and extensible formal framework

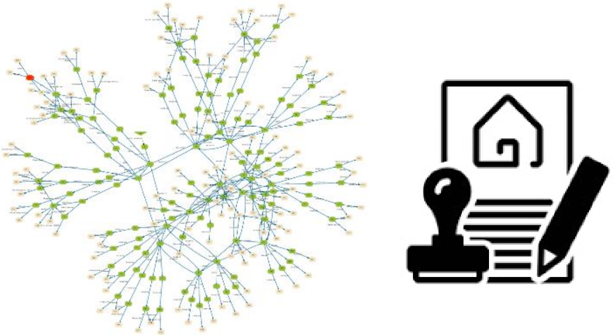
Challenges

- 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

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 σ to be a valuation of all variables in Vars, including

$\text{tag} : \{ \text{normal}, \text{gate}, \text{privileged} \}$

$\text{mem} : (\text{MEM} \times \text{M\_TAG} \times \text{M\_TYPE} \times \text{M\_PRIV} \times \text{M\_INTEGRITY})$

$\text{regs} = \text{addr\_regs} \cup \text{bit\_regs}$

- $\text{addr\_regs} : (\text{MEM} \times \text{R\_PRIV})$, e.g. PC, SP, CR3, TTBR
- $\text{bit\_regs} : (\text{CONTROL\_BIT} \times \text{R\_PRIV})$, e.g. CR0, TTBCR

$\text{flags} : \{\text{interrupt\_flag}, \ldots \}$

Formal Framework: Privilege-Centric Model

The transition function δ will be like:

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

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

e.g. $\psi_{write} \doteq control\_bit(wp) = FALSE \vee write \in m\_priv(m)$

Formal Framework: Security Properties

Security invariants for privilege separation:

I1. $tag = privileged \Rightarrow MEM\_PRIV\_DIFF \subseteq MEM\_PRIV\_SET$

I2. $tag = normal \Rightarrow MEM\_PRIV\_SET \cap MEM\_PRIV\_DIFF = \emptyset$

I3. $tag = privileged \Rightarrow REG\_PRIV\_DIFF \subseteq REG\_PRIV\_SET$

I4. $tag = normal \Rightarrow REG\_PRIV\_SET \cap REG\_PRIV\_DIFF = \emptyset$

I5. $m\_tag(m) = privileged \Rightarrow m\_integ(m) = TRUE$

Formal Framework: Security Properties

Security contracts for the privileged domain:

C1. $tag = privileged \wedge write(m) \Rightarrow execute \notin m\_priv(m)$

C2. $tag = privileged \wedge write(m) \Rightarrow m\_tag(m) \neq normal$

C3. $tag = privileged \wedge execute(m) \Rightarrow m\_tag(m) \neq normal$

Instantiation

B-method

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

```
MACHINE Untitled
SETS
 ID={aa,bb}
CONSTANTS iv
PROPERTIES
 iv:ID
VARIABLES xx
INVARIANT
 xx:ID
INITIALISATION xx:=iv
OPERATIONS
  Set(yy) = PRE yy:ID THEN xx:= yy END
END
```

The abstract machine structure of B-method

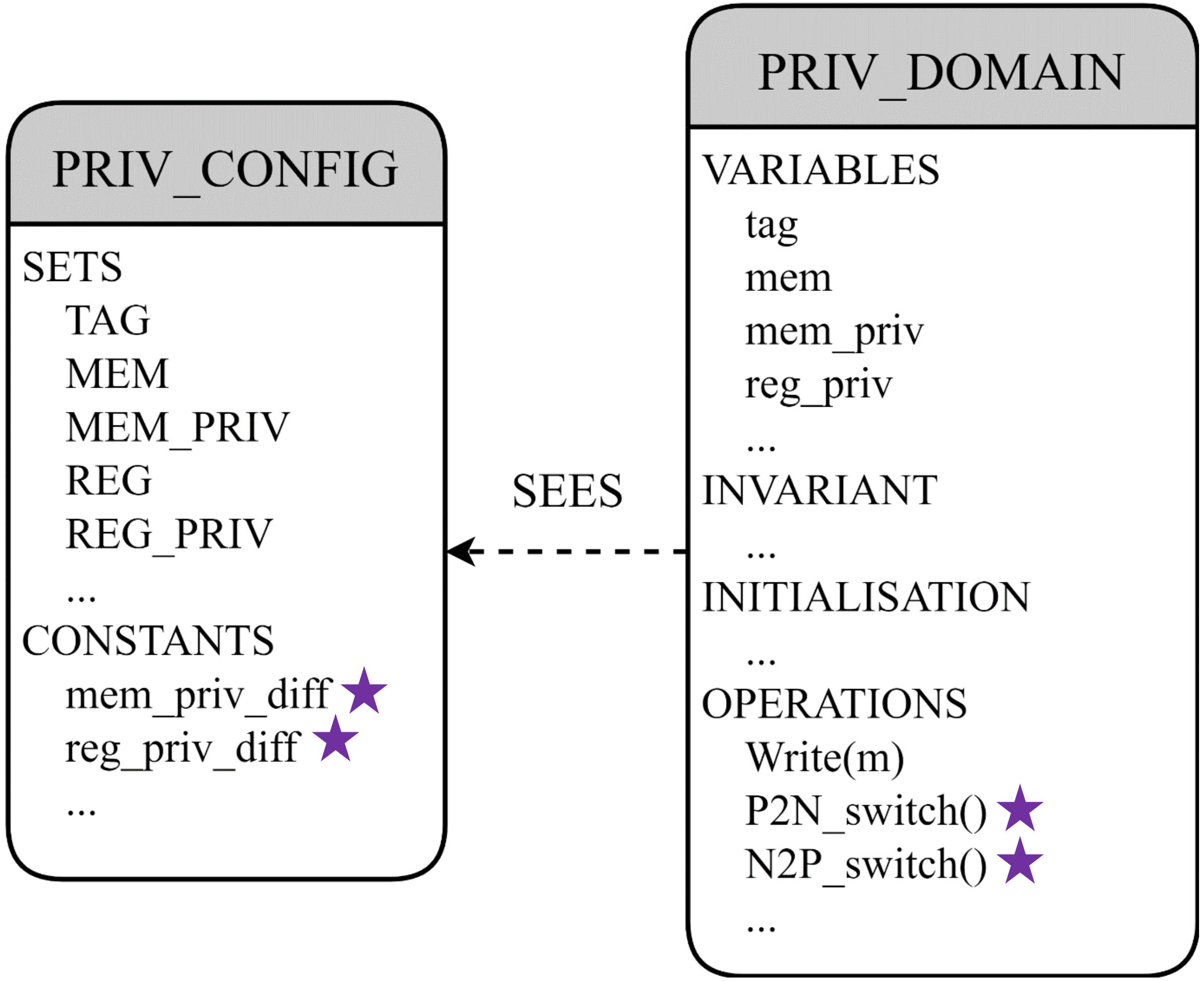
Instantiation

From developers

- Privilege differences
- Privilege switch (ψ_{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

Abstract Machine	States	Transitions	Time(s)	Memory(MB)	Deadlock	Invariant Violations
Nested Kernel	2554	4886	2.559	174.039	/	/
Hilps	44546	105218	47.745	238.102	/	/

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

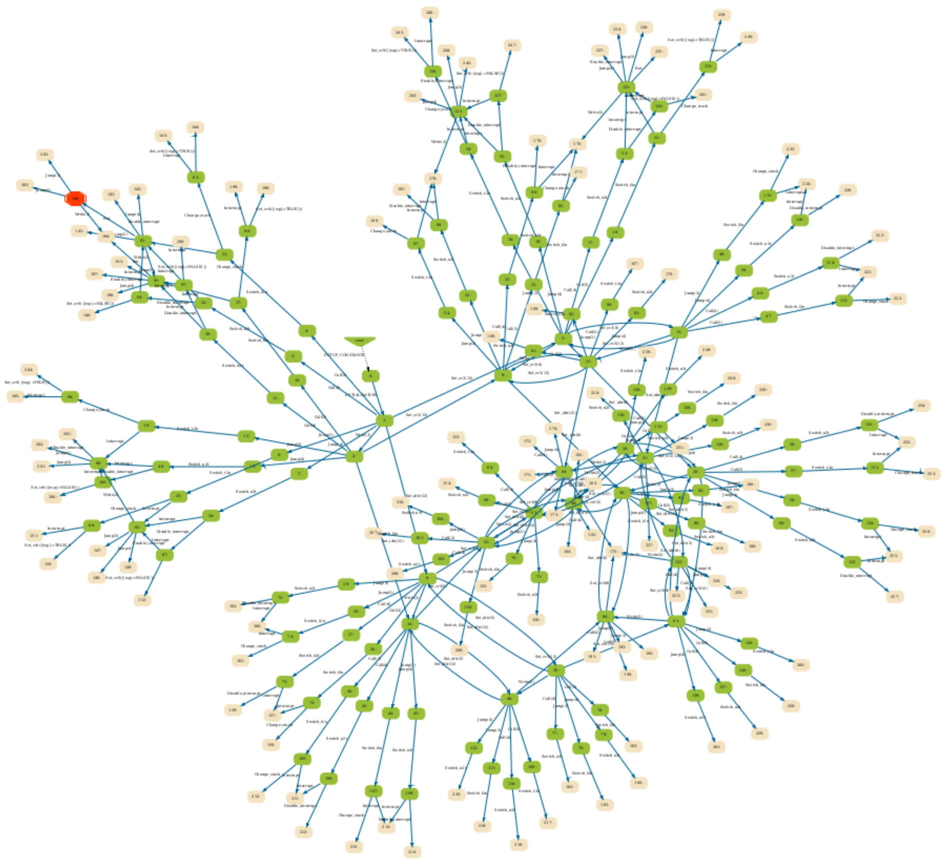
Case Study

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

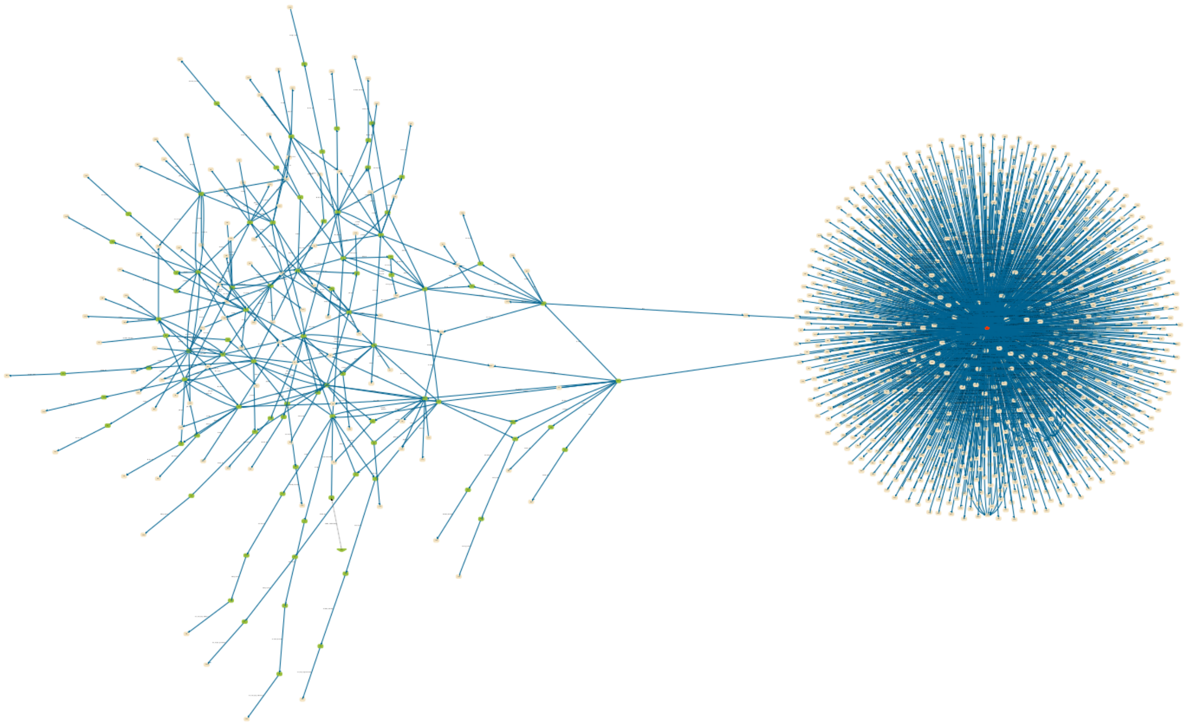
Table 4: Nested Kernel Model Checking Results for Attack Scenarios

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

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