Analyzing The Security of The Cache Side Channel Defences With Attack Graphs

Limin Wang, Ziyuan Zhu, Zhanpeng Wang, Dan Meng

{wanglimin, zhuziyuan, wangzhanpeng, mengdan}@iie.ac.cn

Institute of Information Engineering
Chinese Academy of Sciences

Outline

- **Introduction**
- **Motivation**
- **Method**
- **Experiments**
- **Conclusions**

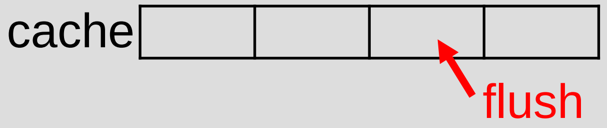
Introduction

Cache side channel attacks on the microarchitecture

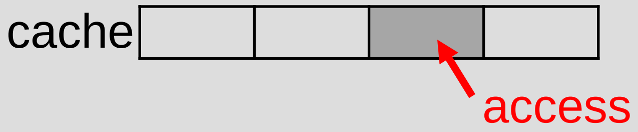
FLUSH+RELOAD[1]	EVICT+TIME[2]
PRIME+PROBE[3]	CACHE COLLISION[4]

Flush+Reload in detail

① Flush cache lines mapped from carefully chosen memory address.



② Wait for the victim to access the flushed address and re-cache the data.



③ The attacker will re-access the chosen memory address, record the time.

Short access time -> victim has accessed
Fast access time -> victim has not access

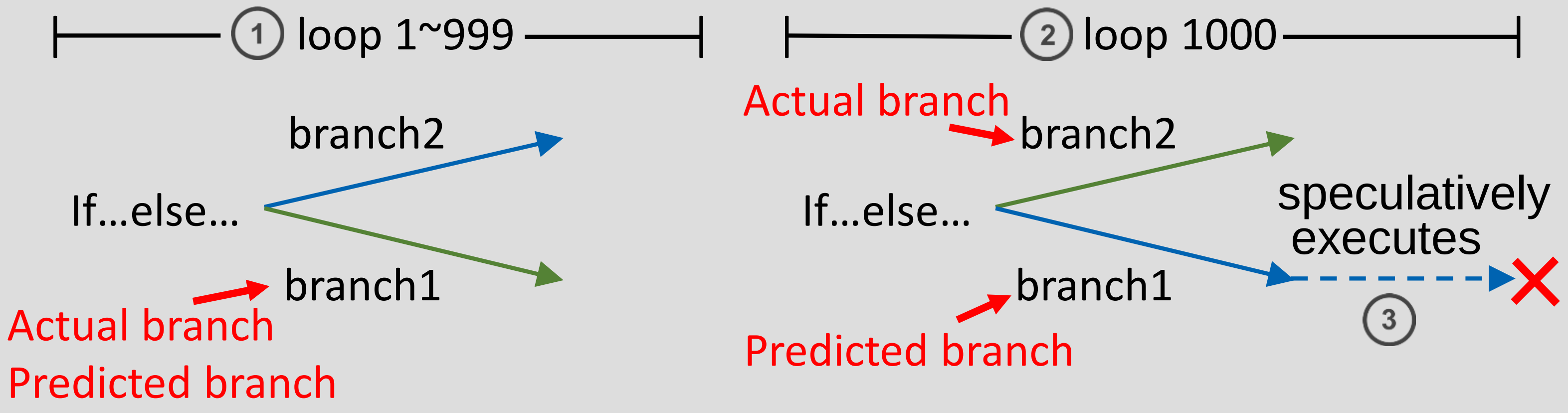
Introduction

Hardware vulnerabilities on the microarchitecture

Spectre[5]	Meltdown[6]

Spectre in detail

① Train the branch predictor, make a wrong branch prediction.
② Exploit it, then the processor speculatively executes the attack program.
③ The attacker accesses the victim's data illegally and make them cached.



Introduction

The increasing risk of cache side channel attacks

FLUSH+RELOAD	EVICT+TIME
PRIME+PROBE	CACHE COLLISION

Hardware vulnerabilities make cache attacks more powerful

① Flush carefully chosen cache lines.
② Waits for the victim to access the flushed address and re-cache the data.
③ Re-access the cache lines, record the time.

Meltdown Spectre

The attacker accesses data illegally and make them cached.

Outline

- Introduction
- **Motivation**
- Method
- Experiments
- Conclusions

Motivation

Analyze the security of the defences

Proposed defences to prevent attacks

$$\begin{tikzcd} \text{Software} \arrow[r] & \text{OS} \arrow[r] & \text{Secure Cache} \arrow[r] & \text{Microarchitecture} \end{tikzcd}$$

Software	OS	Secure Cache	Microarchitecture
Code Obfuscation [7]	KPTI [8]	SP Cache [9]	InvisiSpec [10]

The scope of defences extends from software to hardware

Problems of microarchitecture defences

① Hardware is hard to patch after been published.

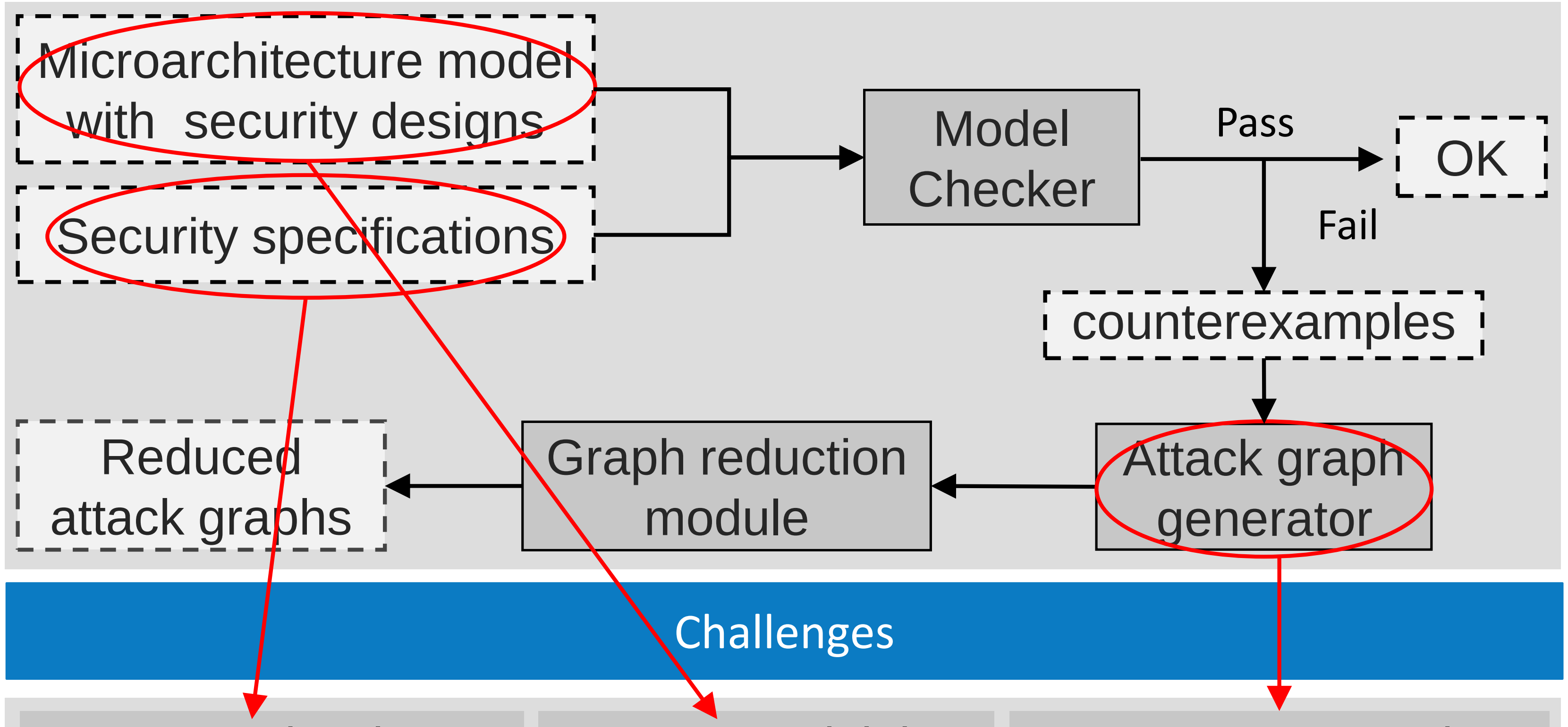
② It is also hard to design perfect defences.

It is necessary to analyze the security of the microarchitecture defences at the early stage of designing the processors[11]

Outline

Method

Proposed Approach: Analyzing defences with attack graph



Challenges

How to develop security specifications for various exploits?	How to model the complex microarchitecture?	How to generate the Attack Graph for cache side channel attacks?

Method

Proposed Approach: Detail

1. How to develop security specifications for various exploits?

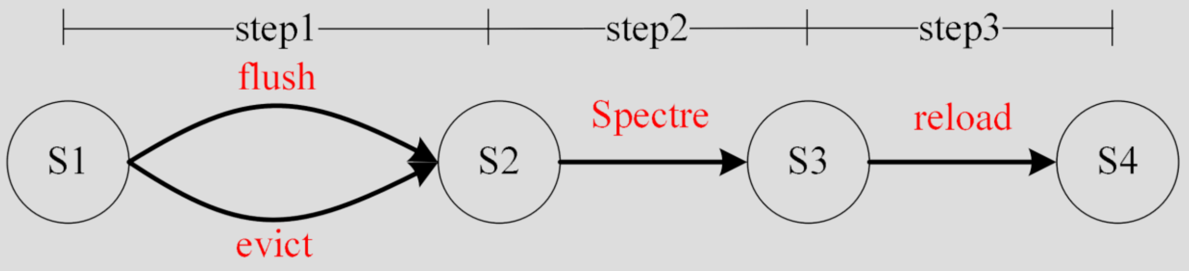
● **Background**:	the features of different exploits
● **Approach**:	expressed as a sequence of states
● **How to**:	the details of our method
● **Advantages**:	the advantages of our method

2. How to model the complex microarchitecture?

3. How to generate the Attack Graph for cache side channel attacks?

Method

1. Security specifications: Background

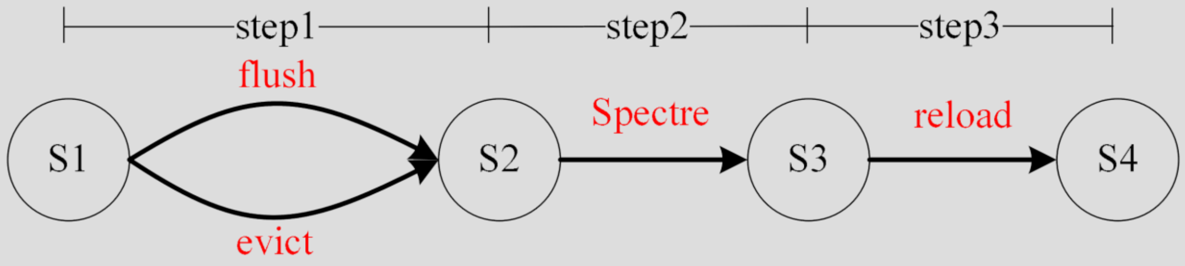


The features of different exploits:

1. The cache attacks can be divided into several steps.
2. Successful attack step => states become what the attacker expects.
3. Different methods to make the system reach the insecure states.
4. The aim of the defences => make the insecure states hard to reach.

Method

1. Security specifications: Our approach



Key points: Even though exploits are increasing rapidly, but the relevant insecure states do not .

How to develop security specifications

Traditional: Methods in each step

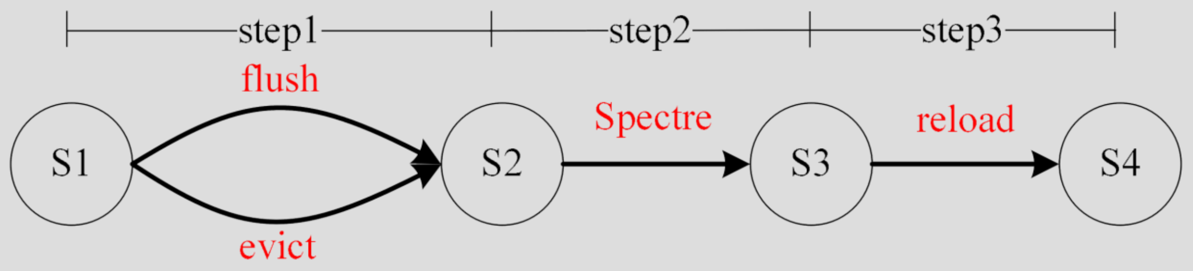
1. Flush->Spectre->Reload 2. Evict->Spectre->Reload

This paper: A sequence of insecure states

1. S1->S2->S3->S4

Method

1. Security specifications: How to



How to develop security specifications for exploits:

1. Divide the attack into several steps manually.
2. Analyze the insecure states.
3. Express the security specification with computation tree logic.

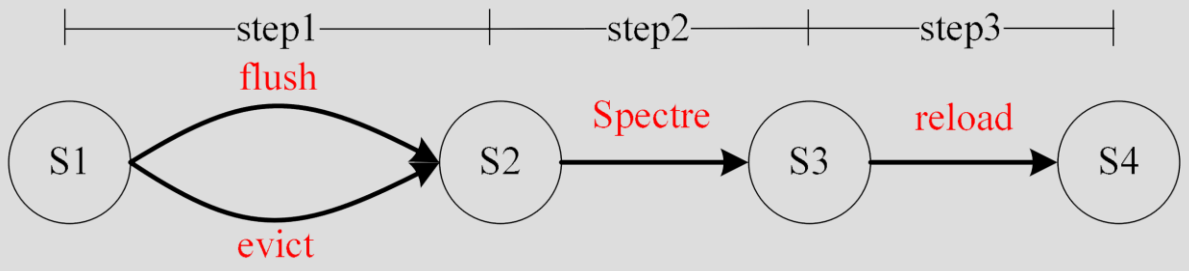
Security Specification： ¬EF(S2 U S3)

Do *NOT E*xist a sequence in the *F*uture

In the sequence, S2 holds *U*ntil S3 is true

Method

1. Security specifications: Advantages



Advantages:

① A security specification is able to represent a class of exploits.

② Can enumerate the known and unknown attack paths that are able to reach these insecure states.

Method

Proposed Approach： Detail

1. How to develop security specifications for various exploits?

2. How to model the complex microarchitecture?

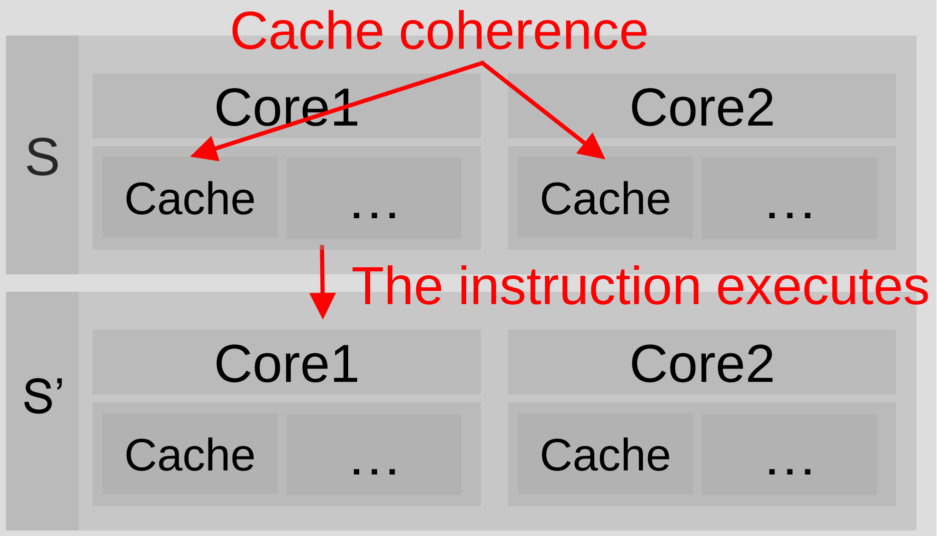
- **Background:** the properties of the microarchitectures
- **Approach:** abstract instruction method
- **How to：** the details of our method
- **Advantages:** the advantages of our method

3. How to generate the Attack Graph for cache side channel attacks?

Method

2. Microarchitecture model: Background

The features of microarchitecture :



① Complex & with lots of properties.

② The states of the properties change:

-- triggered by the execution of instructions.
-- due to the restrictions between different microarchitecture components.

Method

2. Microarchitecture model: Our approach

Abstract instruction model:

Model
Build a state transition model for microarchitecture at the instruction level. ⬇ To express the states change triggered by instructions and restrictions.

The microarchitecture model has to satisfy the security specification

Noted:

Security Specifications
Do not exist a sequence of insecure states.

$$S \xrightarrow[\text{restrictions}]{\text{instructions}} S'$$

satisfy

¬EF(S2 U S3)

Method

2. Microarchitecture model: How to

How to build a model:

M=(S, I, R) S= {A, V, AI, C}

Microarchitecture model
S : a set of microarchitecture states
I : initial states
R : transition relations, $R \subseteq S \times S$

Properties
A: Attacker V: Victim AI: Abstract Instructions C: Microarchitecture Components

$$\{A,\ V,\ AI,\ C\} \xrightarrow{\text{Instructions} \in AI} \{A',\ V',\ AI',\ C'\}$$

Method

2. Microarchitecture model: How to

How model checking works:

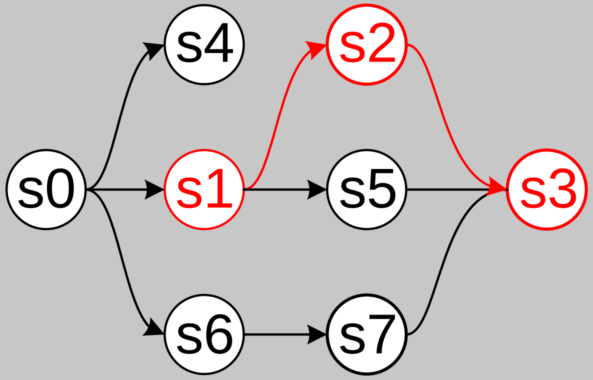
Model:

M=(S, I, R)

I= {s0}

S= {s0, s1, s2, s3, s4, s5, s6, s7}

R= {s0->s1, s0->s4, s0->s6,
s1->s2, s1->s5, s6->s7,
s2->s3, s5->s3, s7->s3}



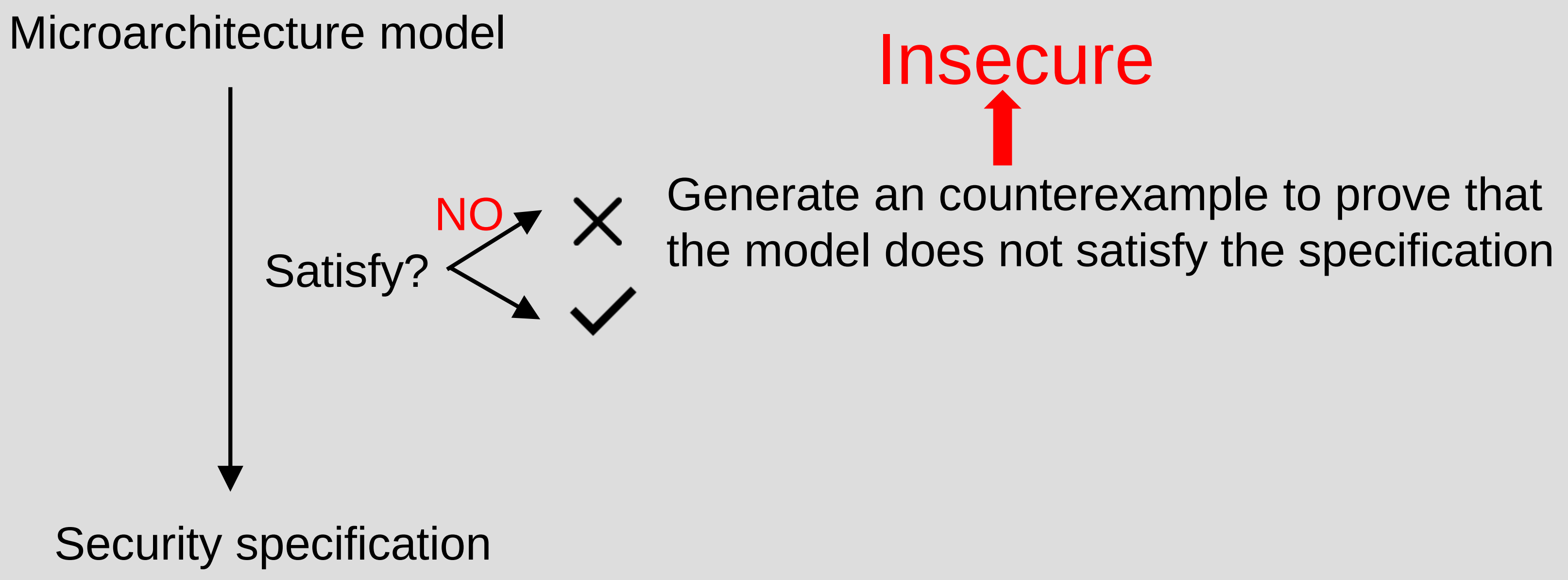
Security Specification: ¬EF (s1->s2->s3)

Counterexample: s0->s1->s2->s3

Method

2. Microarchitecture model: How to

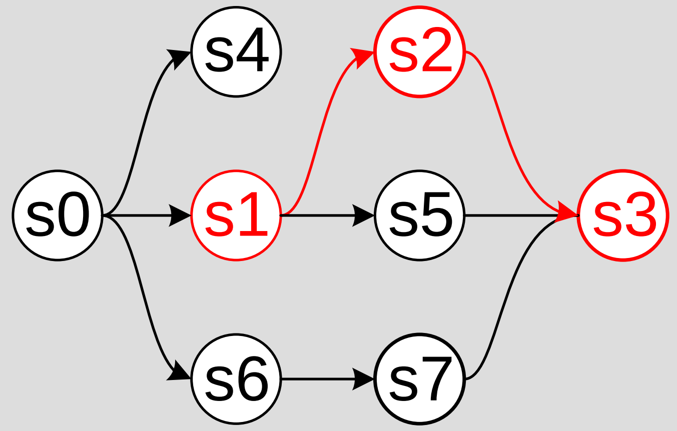
Come back to our method:



Generate an counterexample to prove that the model does not satisfy the specification

Method

2. Microarchitecture model: Advantages



Advantages:

1. Abstract model removes redundant features and Can conveniently express the microarchitecture.

2. The modeling method facilitates the construction of attack graphs.

Method

Proposed Approach： Detail

1. How to develop security specifications for various exploits?

2. How to model the complex microarchitecture?

3. How to generate the Attack Graph for cache attacks?

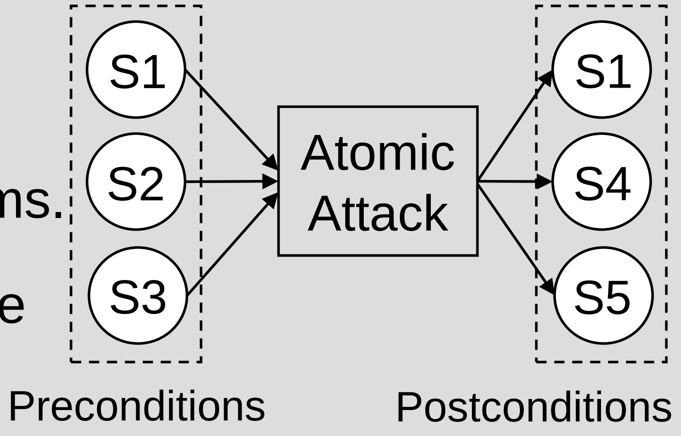
● **Background:**	what is the attack graph
● **Our approach:**	how to generate an attack graph
● **How to:**	the details of our method
● **Advantages:**	the advantages of our method

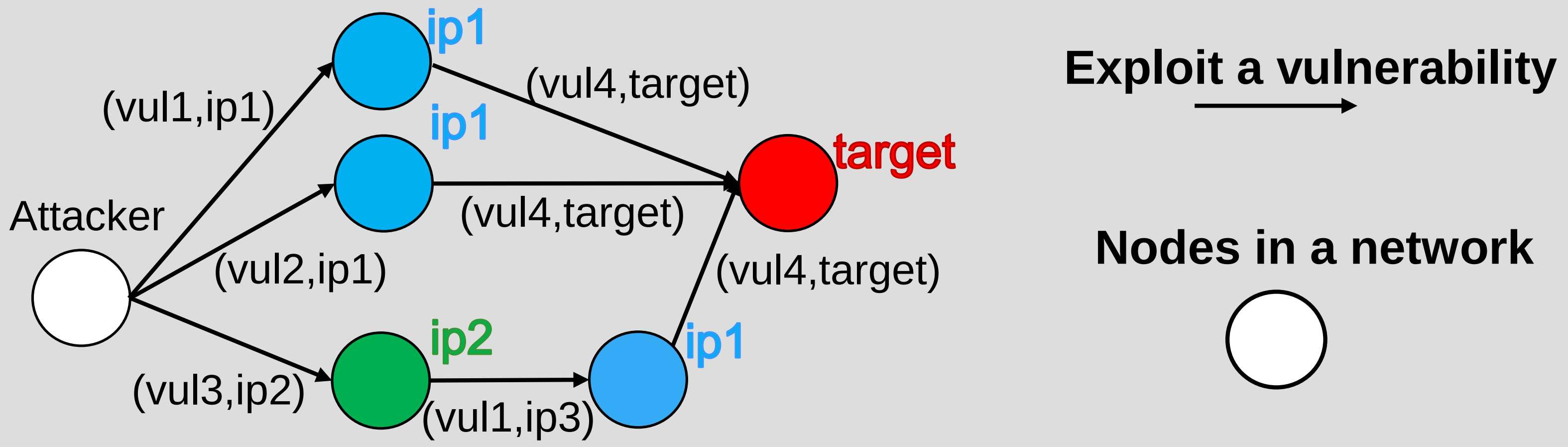
3. Attack Graphs: Background

What is the attack graph:

1. Preconditions: the states have to be true before the atomic attack performs.

2. Postconditions: the states will be true after the atomic attack performs.



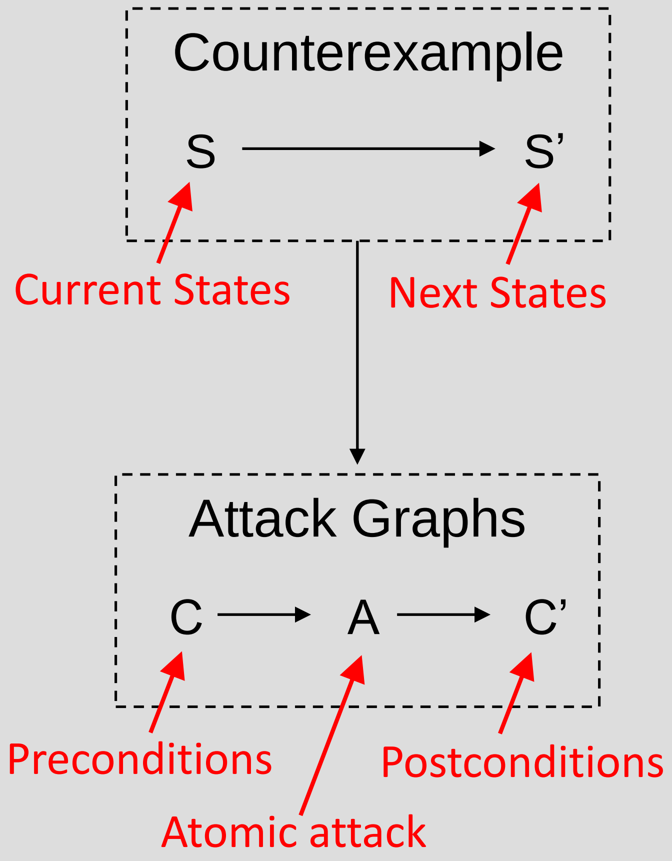


3. Attack Graphs: Our approach

Our approach:

① State transition model=> The counterexample is a sequence of states.

② Divide the counterexample into 3 parts, and transform them to attack graph.

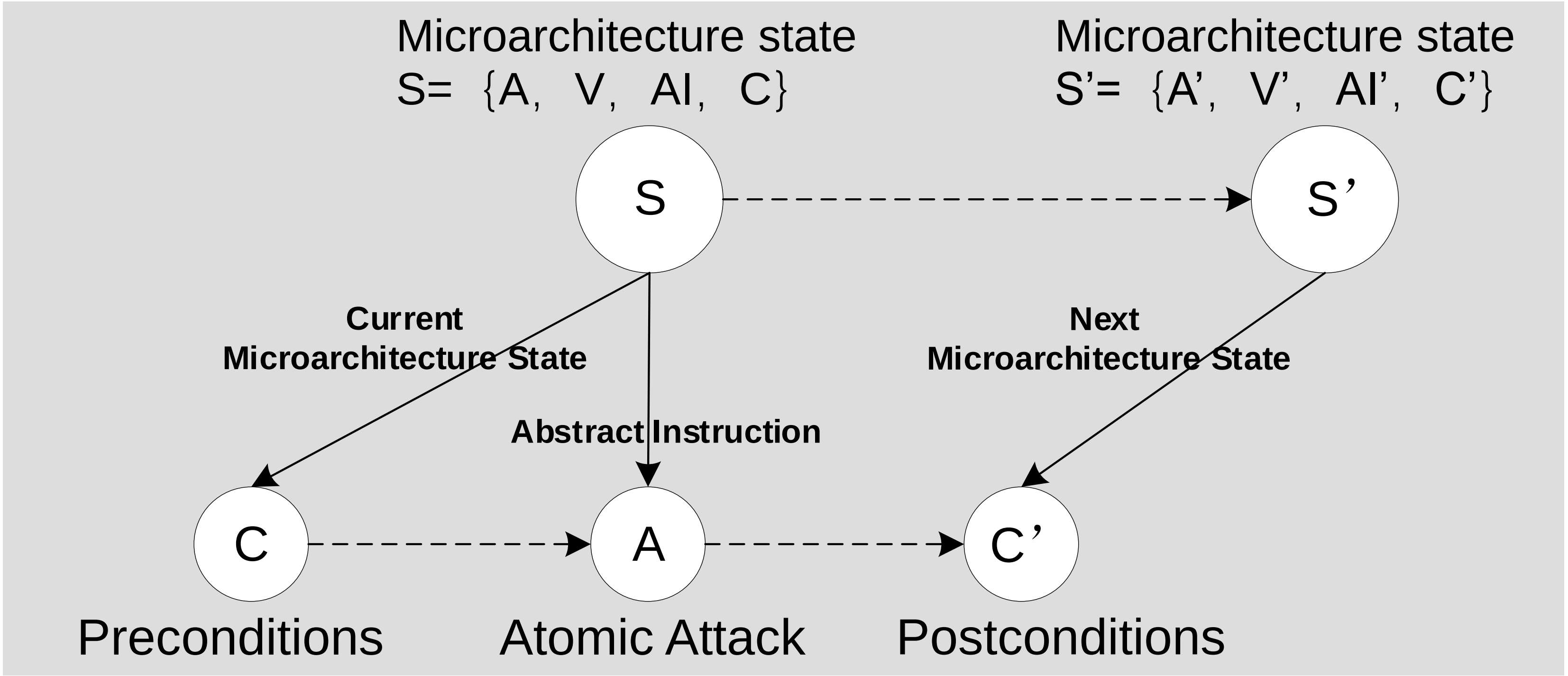


Method

3. Attack Graphs: How to

How to transform:

Properties	A: Attacker V: Victim AI: Abstract Instructions C: Microarchitecture Components

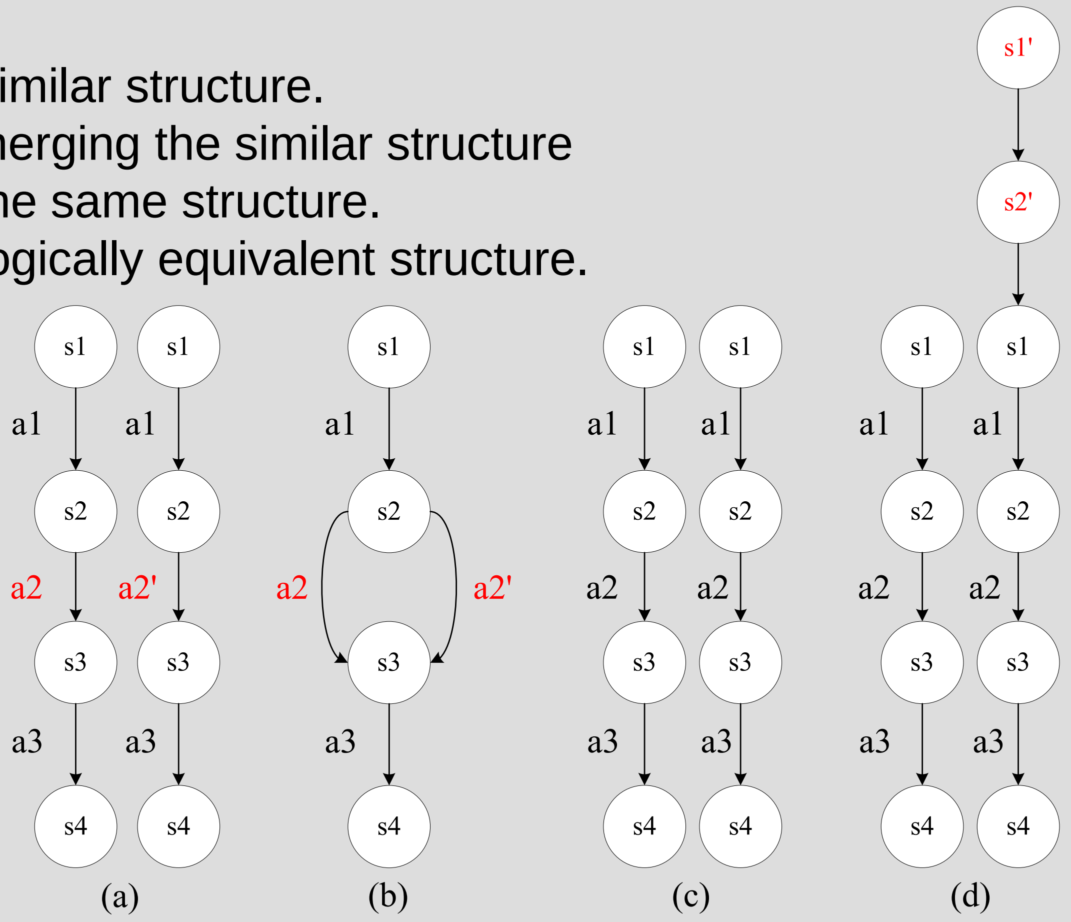


Method

3. Attack Graphs: How to

How to reduce:

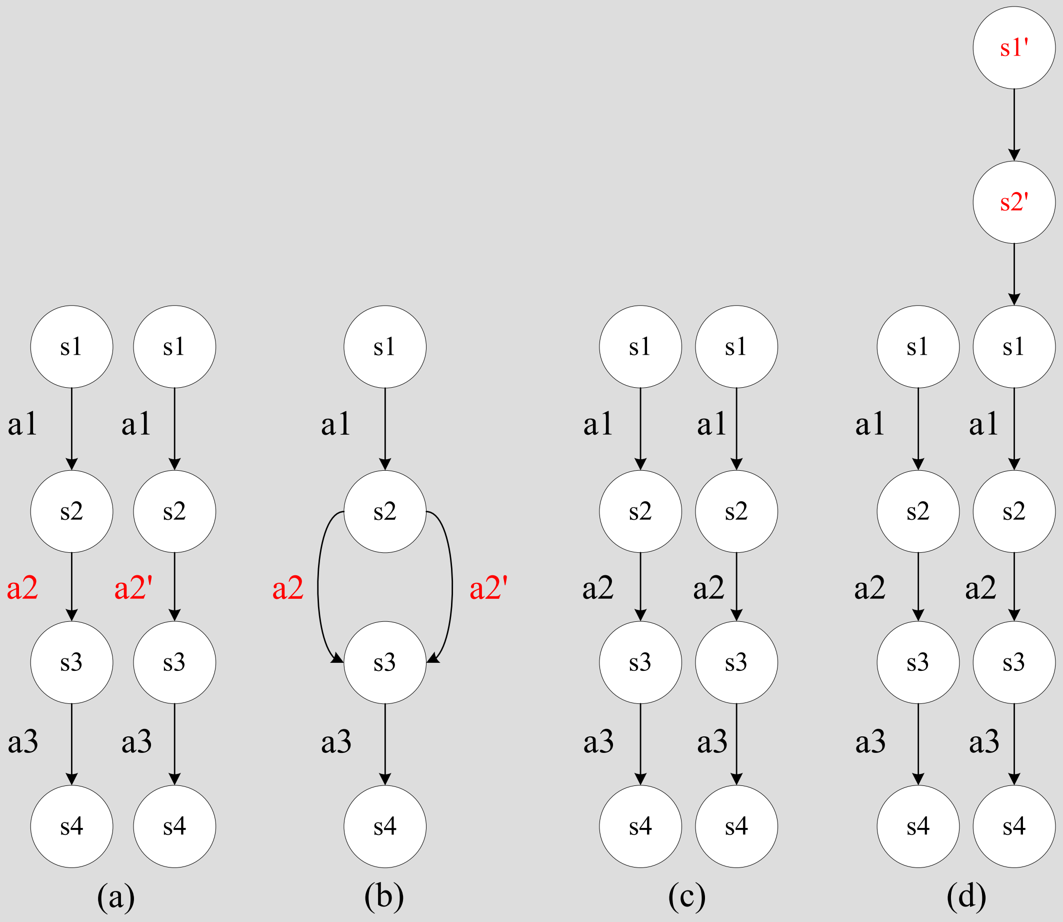
(a) Attack paths with similar structure.
(b) Attack paths that merging the similar structure
(c) Attack paths with the same structure.
(d) Attack paths with logically equivalent structure.



3. Attack Graphs: Advantages

Advantages:

1. High readability.
2. Easy to simplify.

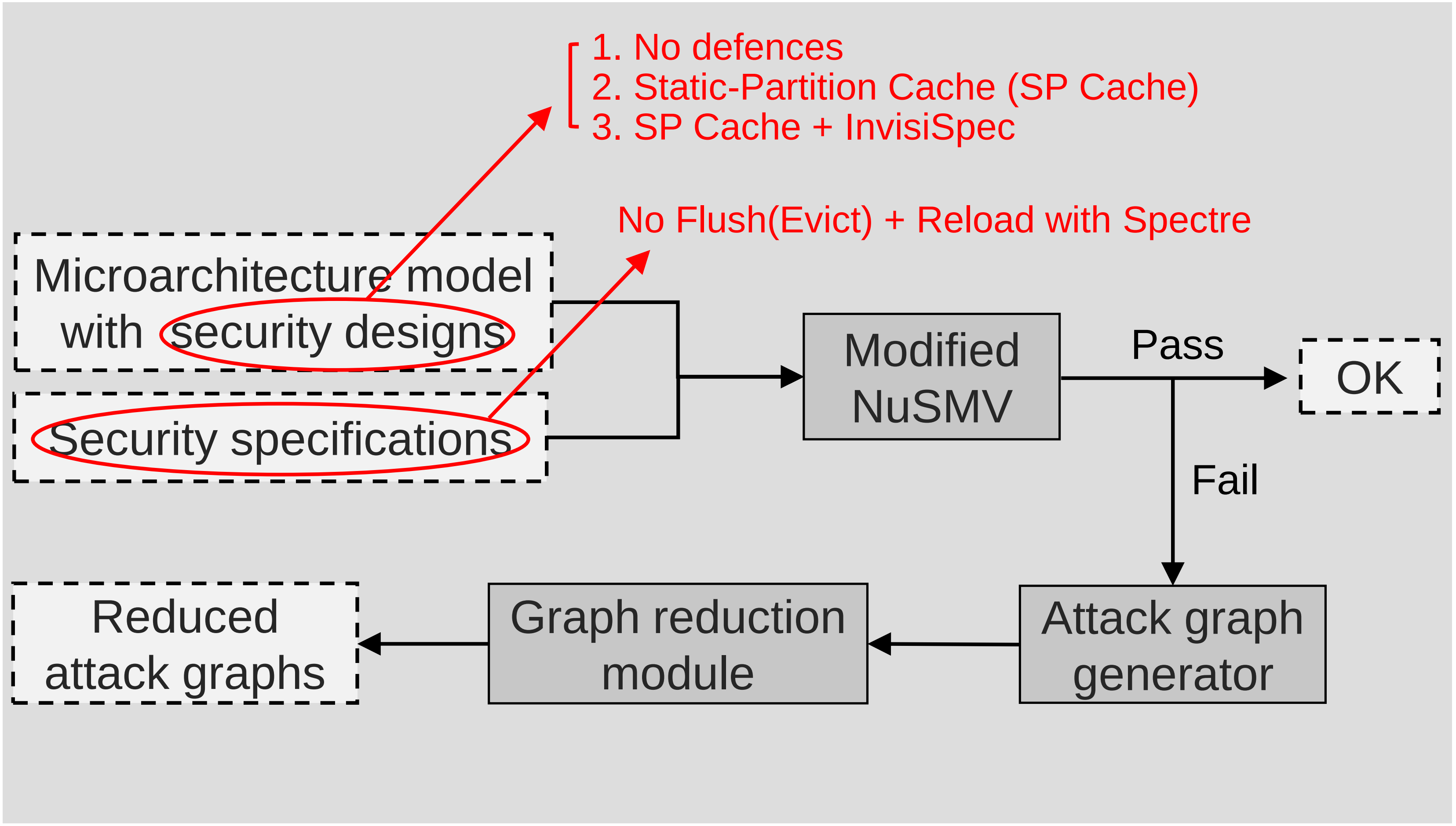


Outline

- Introduction
- Motivation
- Method
- **Experiments**
- Conclusions

Experiments

An Simple Example



Experiments

Microarchitecture model (No Defences)

M=(S, I, R)

Microarchitecture Components & Their Properties

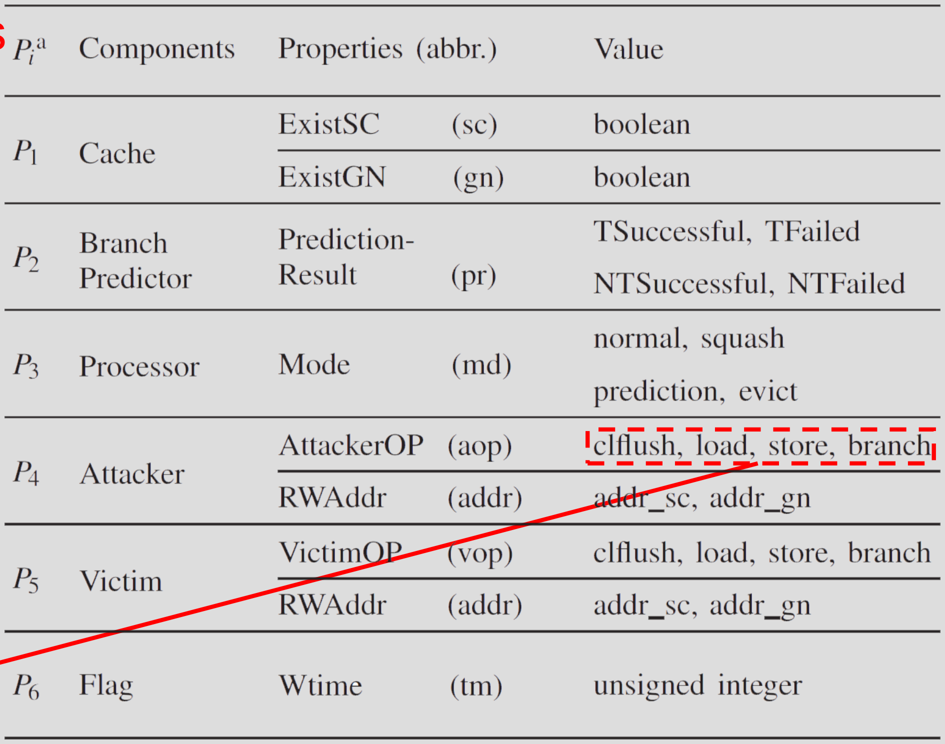
Noted:

More detail the model is,
More accurate the result is,
But more time needed.

Abstract Instructions

clflush load store branch

Table: Selected microarchitecture components and properties



P_i[a]	Components	Properties (abbr.)		Value
P_1	Cache	ExistSC	(sc)	boolean
		ExistGN	(gn)	boolean
P_2	Branch Predictor	Prediction-Result	(pr)	TSuccessful, TFailed NTSuccessful, NTFailed
P_3	Processor	Mode	(md)	normal, squash prediction, evict
P_4	Attacker	AttackerOP	(aop)	clflush, load, store, branch
		RWAddr	(addr)	addr_sc, addr_gn
P_5	Victim	VictimOP	(vop)	clflush, load, store, branch
		RWAddr	(addr)	addr_sc, addr_gn
P_6	Flag	Wtime	(tm)	unsigned integer

Experiments

Microarchitecture model (No Defences)

M=(S, I, R)

State Transition

$$S \xrightarrow{\text{instructions}} S'$$

Load (attacker) =>
attacker's data is in cache =true

Load (victim) =>
victim's data is in cache =true

$$\frac{\begin{cases} gn' = true\,,\ if(\,aop = load\,) \\ sc' = true\,,\ if(\,vop = load\,) \end{cases} ,\ aop' \in A\,,\ vop' \in A\,,\ tm' = 0}{(aop\,,\ vop\,,\ gn\,,\ sc\,,\ tm) \rightarrow (aop'\,,\ vop'\,,\ gn'\,,\ sc'\,,\ tm')}$$

Load

$$\frac{\begin{cases} gn' = true\,,\ if(\,aop = load\,) \\ sc' = true\,,\ if(\,vop = load\,) \end{cases} ,\ aop' \in A\,,\ vop' \in A\,,\ tm' = 0}{(aop\,,\ vop\,,\ gn\,,\ sc\,,\ tm) \rightarrow (aop'\,,\ vop'\,,\ gn'\,,\ sc'\,,\ tm')} \quad \text{load}$$

$$\frac{\begin{cases} gn' = true\,,\ if(\,aop = store\,) \\ sc' = true\,,\ if(\,vop = store\,) \end{cases} ,\ aop' \in A\,,\ vop' \in A\,, \quad \begin{cases} tm' = 0 \wedge md' = evict \wedge sc' = false\,;\ if(\,tm = n-1 \wedge aop = store\,) \\ tm' = tm + 1 \qquad ;\ if(\,tm < n-1 \wedge aop = store\,) \end{cases}}{(aop, vop, gn, sc, tm) \rightarrow (aop', vop', gn', sc', tm')} \quad \text{store}$$

$$\frac{aop = branch \vee vop = branch\,,\ md' = prediction\,,\ tm' = 0\,, \quad pr' \in \begin{Bmatrix} TSuccessfull & TFailed \\ NTSuccessfull & NTFailed \end{Bmatrix} ,\ vop' \in A\,,\ aop' \in A}{(aop\,,\ vop\,,\ md\,,\ pr\,,\ tm) \rightarrow (aop'\,,\ vop'\,,\ md'\,,\ pr'\,,\ tm'))} \quad \text{branch}$$

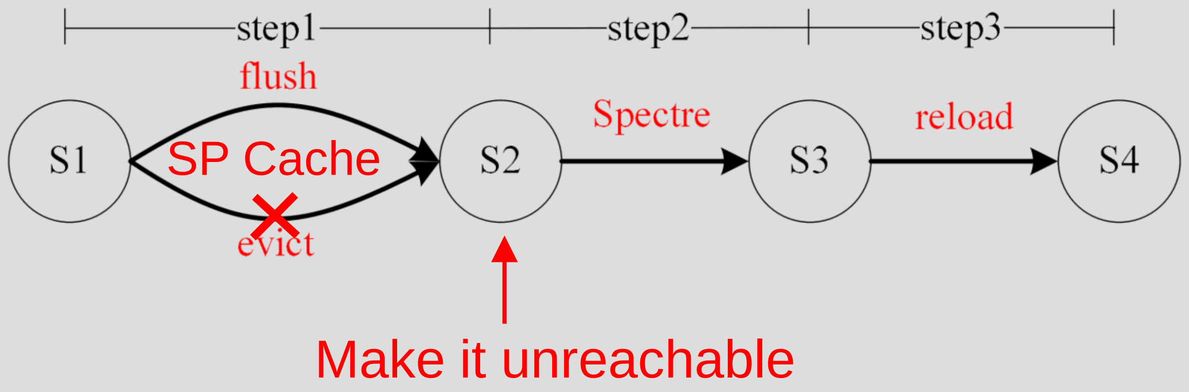
$$\frac{\begin{cases} md' = squash\,,\ if(\,md = prediction\ \wedge pr \in \begin{Bmatrix} TFailed \\ NTFailed \end{Bmatrix}) \\ md' = normal\,,\ if(\,md = prediction \wedge pr \in \begin{Bmatrix} TSuccessful \\ NTSuccessful \end{Bmatrix}) \end{cases}}{(md) \rightarrow (md')} \quad \text{branch}$$

$$\frac{\begin{cases} sc' = false\,;\ if(\,addr = addr\_sc\,) \\ gn' = false\,;\ if(\,addr = addr\_gn\,) \end{cases} ,\ aop' \in A\,, \quad aop = clflush \vee vop = clflush\,,\ tm' = 0\,,\ vop' \in A}{(aop\,,\ vop\,,\ sc\,,\ gn\,,\ tm) \rightarrow (aop'\,,\ vop'\,,\ sc'\,,\ gn'\,,\ tm'))} \quad \text{clflush}$$

Figure: State transition for abstract instructions

Experiments

Microarchitecture model (SP Cache)



Static-Partition Cache (SP Cache)

-- Statically separates the cache for the victim and the attacker.

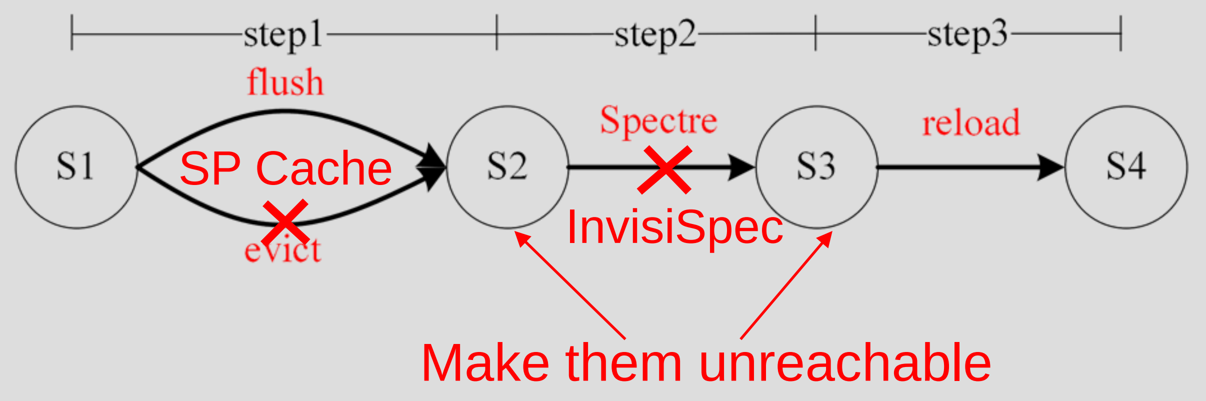
-- Attacker cannot evict victim’s cache lines.

-- The state transition about Evict strategy will be deleted from the R.

M=(S, I, R)

Experiments

Microarchitecture model (SP Cache + InvisiSpec)



InvisiSpec

-- InvisiSpec loads data into a new Speculative Buffer before committing the result until the speculative load is finally safe.

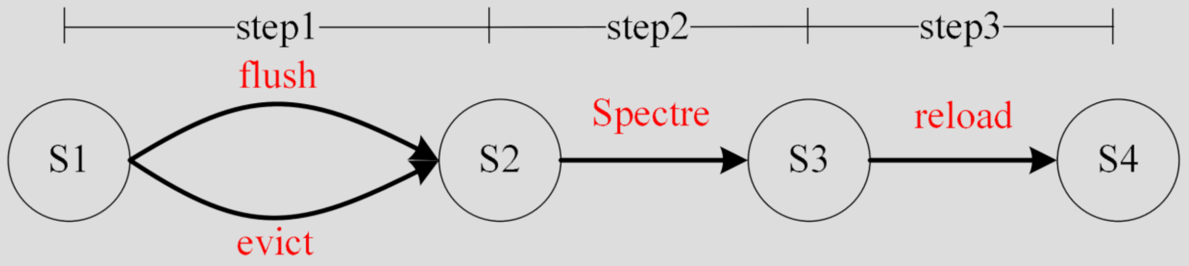
-- The state transition about the branch will be modified to the mechanism InvisiSpec describes.

M=(S, I, R)

Experiments

Security Specification

Flush (Evict) + Reload with Spectre



A sequence of insecure states

Security Specification (informal): **¬*EF*(*S2* U *S3*)**

Security Specification (formal): **¬*EF***(*E*[*sc* = *false* ***U***((*md* = *squash*)*EX*(*sc* = *true*))])

Do *NOT* *E*xist a sequence that S2 holds *U*ntil S3 is true in the *F*uture

Experiments

Results

Microarchitecture model
1. With no defences
2. With SP Cache
3. With SP Cache + InvisiSpec

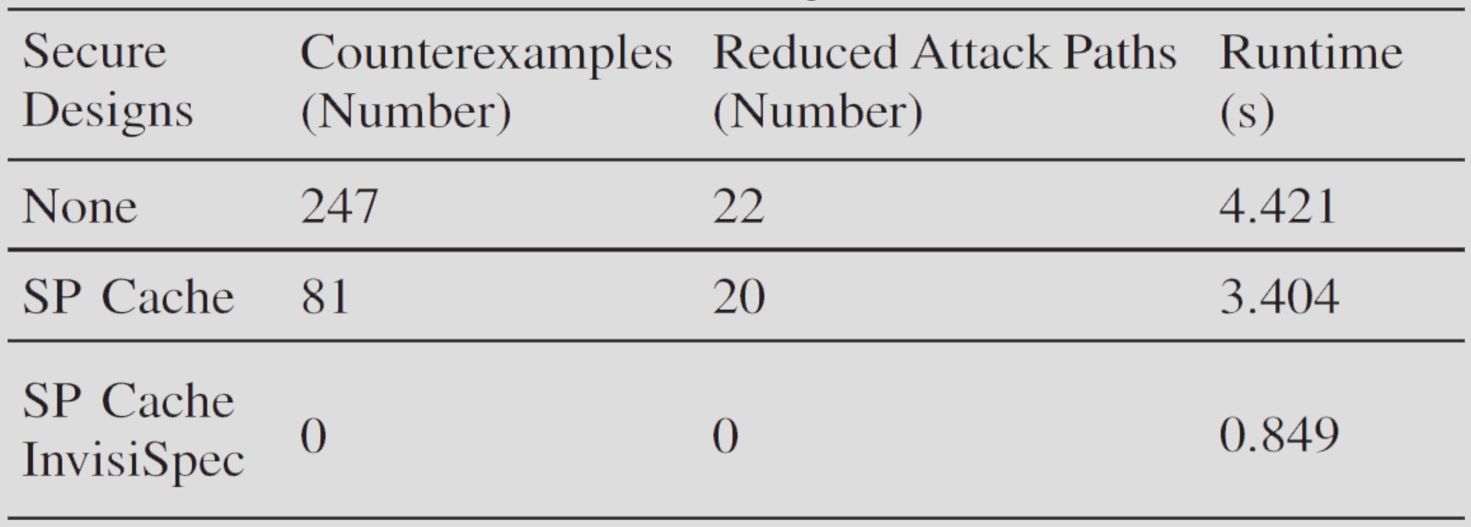
Model Checking

$\longrightarrow$

Satisfy?

Security specification
¬EF(S2 U S3)
(Flush + Reload with Spectre)
(Evict + Reload with Spectre)

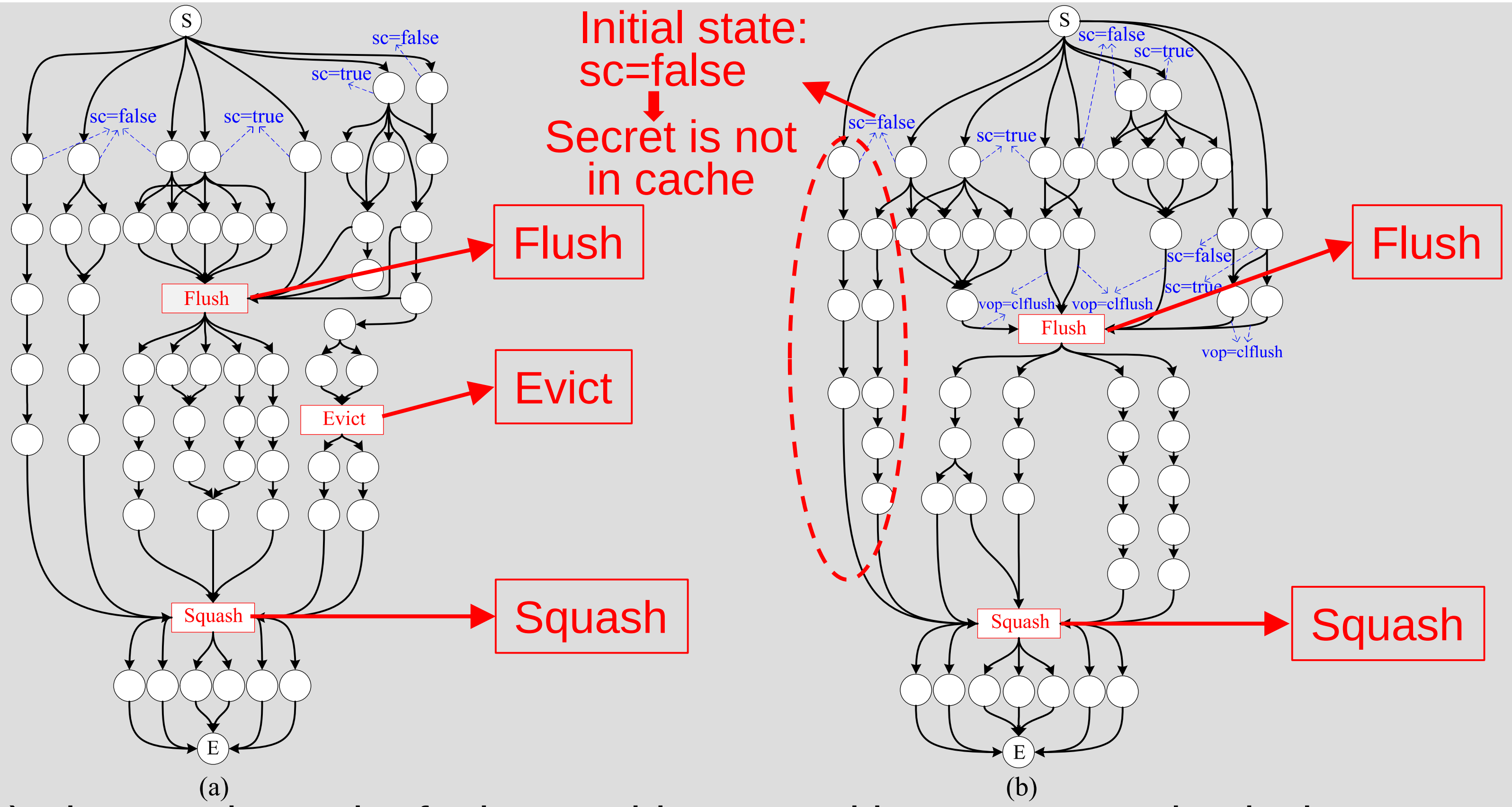
Table : Results of model checking



Secure Designs	Counterexamples (Number)	Reduced Attack Paths (Number)	Runtime (s)
None	247	22	4.421
SP Cache	81	20	3.404
SP Cache InvisiSpec	0	0	0.849

Experiments

Attack Graphs



(a) The attack graph of micro-architecture without any security designs
(b) The attack graph of micro-architecture with SP Cache

Outline

- Introduction
- Motivation
- Method
- Experiments
- **Conclusions**

Conclusions

Conclusions:

1. Use instruction abstract method to conveniently
   model the microarchitecture.

2. Use the sequence of insecure states to express
   the security specification.

3. Proposes a novel use of the attack graph technology to visualize
   the cache side-channel attack path.

Limitions：

1. State space explosion problem.

2. The attack that do not violate the security
   specifications will not be identified.

Reference

[1] Yarom, Yuval, and Katrina Falkner. "FLUSH+ RELOAD: a high resolution, low noise, L3 cache side-channel attack." *23rd {USENIX} Security Symposium ({USENIX} Security 14)*. 2014.

[2] E. Tromer, D. A. Osvik, and A. Shamir, “Efficient Cache Attacks on AES, and Countermeasures,” J Cryptol, vol. 23, no. 1, pp. 37–71, Jan. 2010.

[3] M. Kayaalp, N. Abu-Ghazaleh, D. Ponomarev, and A. Jaleel, “A high-resolution side-channel attack on last-level cache,” in Proceedings of the 53rd Annual Design Automation Conference on - DAC ’16, Austin, Texas, 2016, pp. 1–6.

[4] J. Bonneau and I. Mironov, “Cache-Collision Timing Attacks Against AES,” in Cryptographic Hardware and Embedded Systems - CHES 2006, vol. 4249, L. Goubin and M. Matsui, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 201–215.

[5] P. Kocher et al., “Spectre attacks: Exploiting speculative execution,” in 2019 IEEE symposium on security and privacy (SP), 2019, pp. 1–19.

Reference

[6] M. Lipp et al., “Meltdown: Reading kernel memory from user space,” in 27th {USENIX} security symposium ({USENIX} security 18), 2018, pp. 973–990.

[7] B. Köpf, L. Mauborgne, and M. Ochoa, “Automatic Quantification of Cache Side-Channels,” in Computer Aided Verification, vol. 7358, P. Madhusudan and S. A. Seshia, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 564–580.

[8] L. Müller, “KPTI a Mitigation Method against Meltdown,” Advanced Microkernel Operating Systems, p. 41, 2018.

[9] D. Page, “Partitioned cache architecture as a side-channel defence mechanism.”

[10] M. Yan, J. Choi, D. Skarlatos, A. Morrison, C. W. Fletcher, and J. Torrellas, “InvisiSpec: Making Speculative Execution Invisible in the Cache Hierarchy,” p. 14.

[11] A. T. Markettos, R. N. M. Watson, S. W. Moore, P. Sewell, and P. G. Neumann, “Through computer architecture, darkly,” Commun. ACM, vol. 62, no. 6, pp. 25–27, May 2019.

The End

Thank you!

Any questions?