Verifying microarchitectural security guarantees using leakage contracts

Zilong Wang, Hoang Nguyen and Marco Guarnieri

IMDEA software Institute

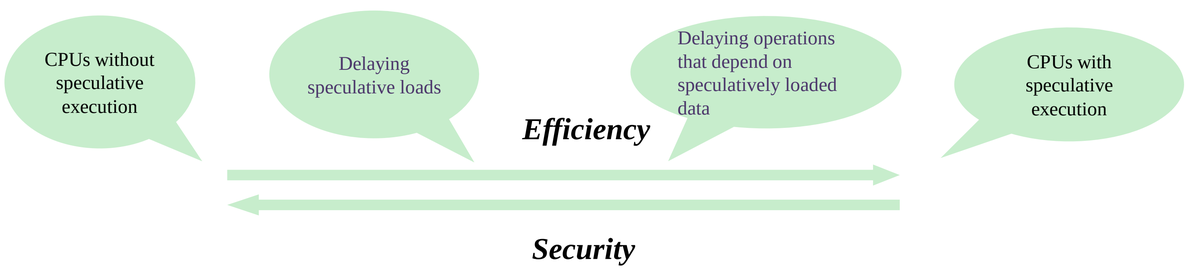
1. What is the challenge?

Efficiency VS. Security

- To avoid expensive pipeline stalls, modern CPUs will speculatively execute corresponding instructions
- Adversary-crafted sequences of transient instructions can access and then transmit sensitive program data over microarchitectural covert channels

```
1  if (y < size_A)
2    x = A[y];
3    temp &= B[x * 64];
```

Modern CPUs will speculatively access even is out of bound!!!!



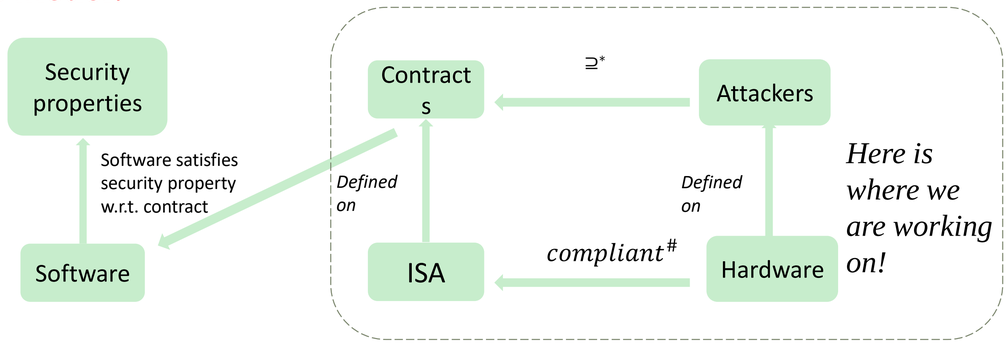
Intuition: ***more defensive*** *mechanisms are* ***less efficient*** *but can securely execute a* ***larger class*** *of programs, while more permissive mechanisms offer* ***more performance*** *but require* ***more defensive*** *programming.*

2. What is a leakage contract?

- Captures the security guarantee of the hardware
- Providing a basis for software to run securely on a specific hardware

The leakage contract is a ***bridge*** across the software and hardware

The model:



- $Contract$: ISA extended with observations
- $Hardware$: Formal model of processor
- Traces $CTR(p,s)$: Sequence of observations
- Traces $HW(p,s)$: Sequence of μarchitectural

* Hardware HW satisfies contract CTR if for all programs p and program states s and s':

$$CTR(p,s) = CTR(p,s') \Rightarrow HW(p,s) = HW(p,s')$$

* $\supseteq$ is defined as leakage order. $A \supseteq B$ means B leaks less than A.

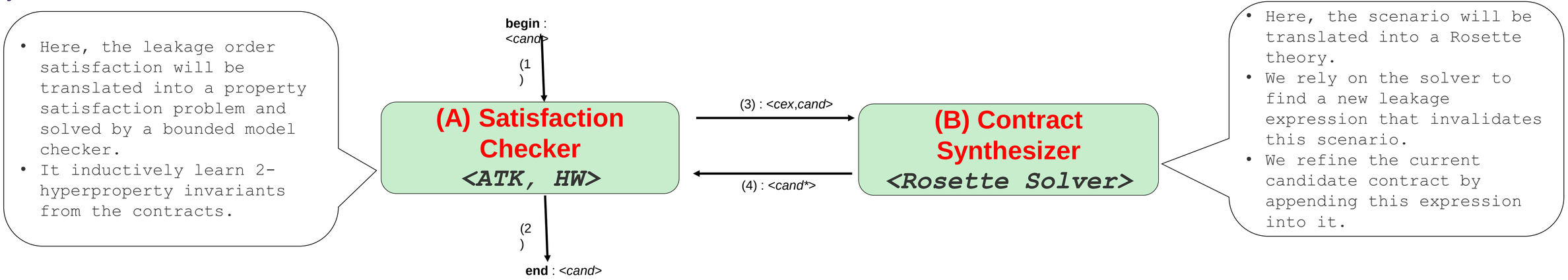
\# *compliance* means a hardware HW correctly implements an architecture ISA.

3. Synthesizing and verifying leakage contracts

GOAL ： Given an attacker, we want to automatically learn the weakest leakage contract for the hardware.

- Two main building blocks for our methodology.

1. **(A) Satisfaction Checker:** given a candidate contract $cand$, an attacker ATK and a hardware design HW, it checks if the $cand$ and ATK satisfy the leakage order
2. **(B) Contract Synthesizer:** is built on top of Rosette solver, it takes a counter-example cex and the candidate contract $cand$ as inputs and synthesize a new $cand^*$ such that it invalidates cex.



- We propose the following counter-example guided methodology to capture hardware leakage contract.

1. We start with the strongest leakage contract as candidate. We use our Satisfaction Checker to verify the candidate.
2. If the candidate is satisfied, then it is returned as the leakage contract of the given hardware.
3. Otherwise, the checker returns scenario where it is unsatisfied. The scenario and candidate are forwarded to our Contract Synthesizer to refine the candidate contract such that it invalidates the scenario.
4. The *cand* is returned to the Checker for the next iteration.

4. What do we already have?

- The satisfaction checker can ***inductively check*** the property which can provide more convinced results compared to methodology based on bounded-model checker.
- We applied the satisfaction checker to some simple CPUs, including defining appropriate contracts, attackers and verifying the leakage order between them.

5. What do we still need to do?

- ***Languages: Improving*** *our formal language to express leakage. The language must be at the same time:*
  - ***expressive*** enough to express interesting leakages.
  - ***restrictive*** enough for the **(B) Contract Synthesizer** to search for sensible solutions.
- ***Applicability: Testing*** *newly defined leakage contracts on various hardware designs.*
- ***Optimization:***
  - **(A) Satisfaction Checker**: ***optimizing*** for checking large scale CPUs.
  - **(B) Contract Synthesizer**: ***providing*** *heuristics (*backtracking, counterexample selection, spurious counterexamples detection etc.) to our synthesize algorithm.

Contact us: *zilong.wang, hoang.nguyen, marco.guarnieri@imdea.org*