Reinforcement Learning for Automated Exploration and Detection of Cache-Timing Attacks in CPS Hardware

Mulong Luo

PhD candidate, Cornell University

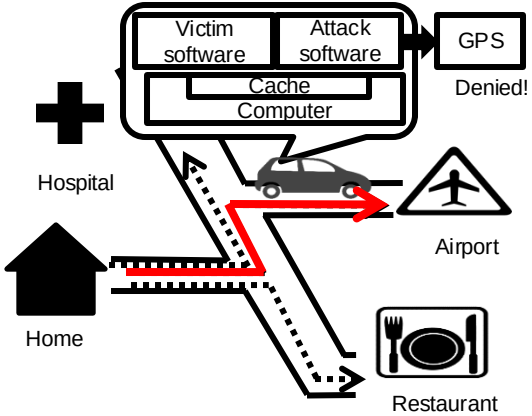
Incoming postdoctoral researcher, UT Austin

July 9th, DACPS workshop, 2023

HW Affects CPS Security

- CPS contains HW Parts
  - E.g., Micro-controller, micro-processor, DRAM
- HW vulnerabilities affect CPS
  - Cache side channels
  - Fault injection attacks
  - Row-hammer attacks
- CPS safety and privacy may be violated due to HW security issues
  - Side channel for tracking autonomous vehicles
  - Interrupt injection for manipulating robotic vehicles

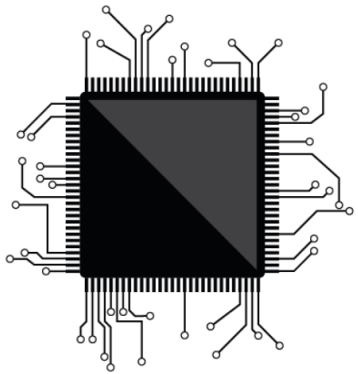
Meltdown/Spectre



Tracking autonomous vehicle with cache timing channel [Luo, USENIX 2020]

Finding HW Vulnerabilities is Hard

- System is too complex
  - laptop processors have ~ 20,000,000,000 transistors
- Undefined system behavior
  - timing of a memory read is unspecified
  - speculative execution that are not committed
- Humans are slow and make mistakes
  - can we use **machine intelligence** to replace them?



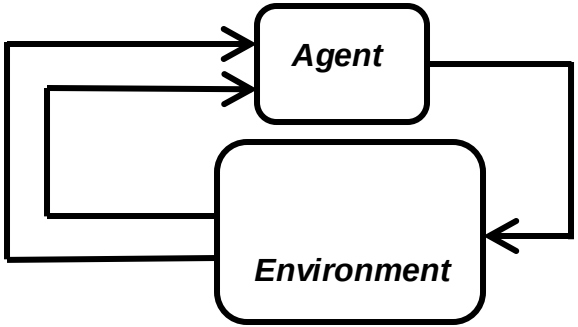
A microprocessor

Meltdown/Spectre

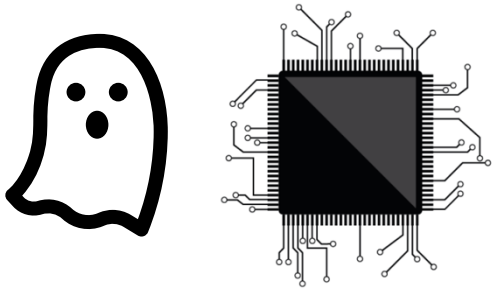
Executive Summary

- Reinforcement learning (RL) can explore cache-timing attacks in processors automatically
  - without explicit specification of processors
  - without knowing existing attack sequences

- RL finds attacks
  - on diverse configurations of caches
  - on real hardware
  - discovers new attack patterns



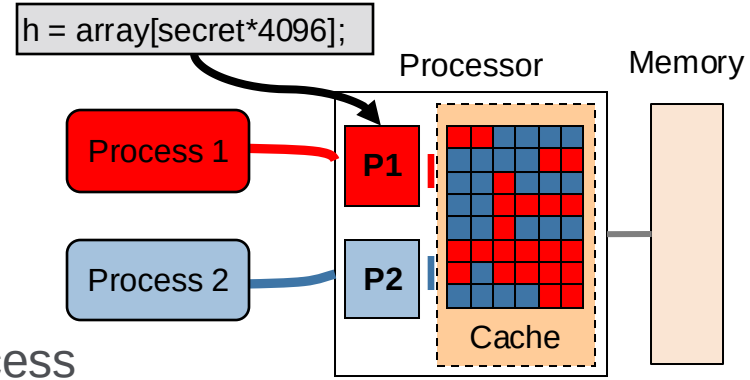
Reinforcement learning scheme

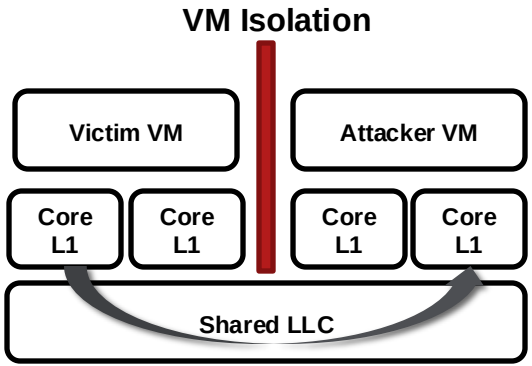


Attack a processor

Cache-Timing Attack: Powerful and Stealthy

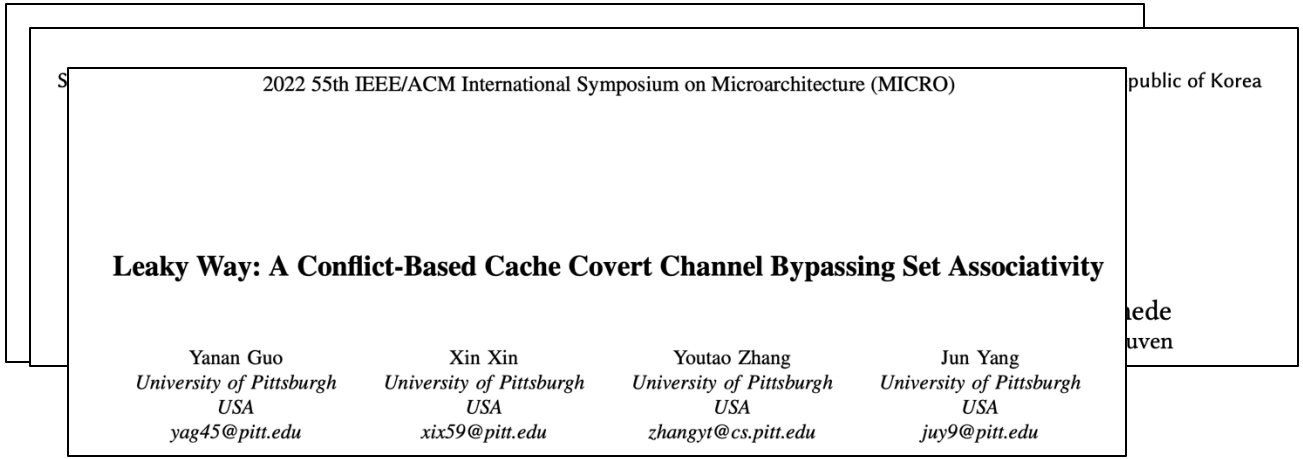
- Mechanism
  - sharing of caches by different processes
  - infer secret by observing cache timing
- Advantages
  - attacker is just a program, no physical access
  - does not violate any OS-level access control
- Leak important assets
  - cryptographic keys
  - VM/browser isolation
  - building blocks for Spectre/Meltdown





Why Finding new Cache-Timing Attack is Hard?

- Cache-timing attack is still developing
  - Traditional: prime+probe, flush+reload, evict+reload, etc
- Finding cache-timing attack is challenging
  - replacement policy complications
  - unknown microarchitectural states
  - …



2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)

Leaky Way: A Conflict-Based Cache Covert Channel Bypassing Set Associativity

Yanan Guo
University of Pittsburgh
USA
yag45@pitt.edu

Xin Xin
University of Pittsburgh
USA
xix59@pitt.edu

Youtao Zhang
University of Pittsburgh
USA
zhangyt@cs.pitt.edu

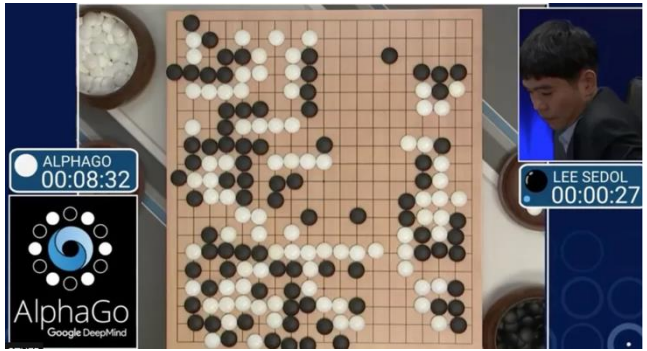
Jun Yang
University of Pittsburgh
USA
juy9@pitt.edu

Existing Tools to Find Vulnerabilities

- Fuzzing
  - pros:
    - require fewer human interventions
  - con:
    - have to deal with large search space
- Formal methods
  - pros:
    - can provably exclude possible vulnerabilities/attacks
  - cons:
    - require RTL of the processor
    - require human to rewrite/implement the formal models

Reinforcement Learning

- RL: a machine learning scheme
  - an agent generates an action sequence
  - maximizes long-term reward



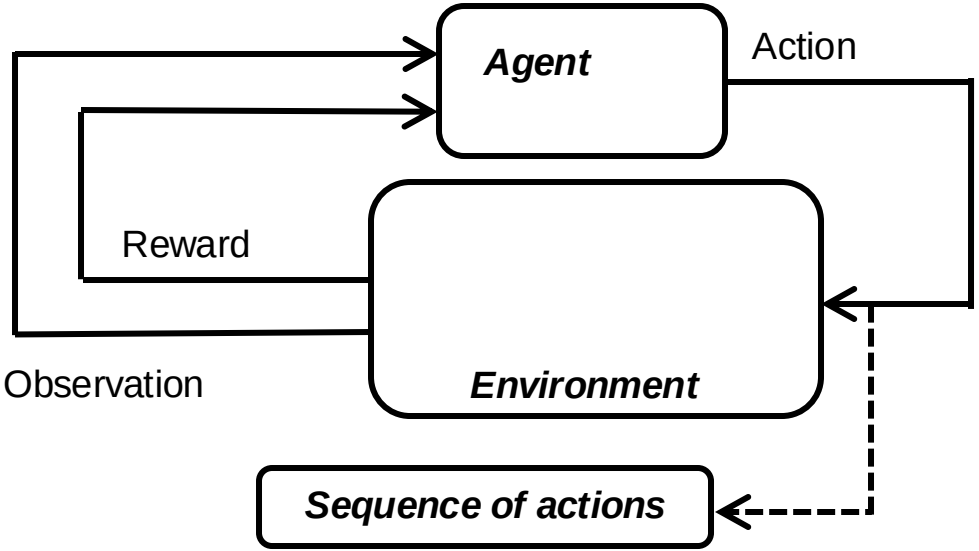
AlphaGo (Source: BBC)

- RL has been applied in game settings to show human-level performance
  - games like Atari (single party)
  - Chess, Go, etc. (two parties)

Atari (source: OpenAI)

Reinforcement Learning

- Key notions
  - agent
  - environment (env)
  - action
  - observation
  - reward



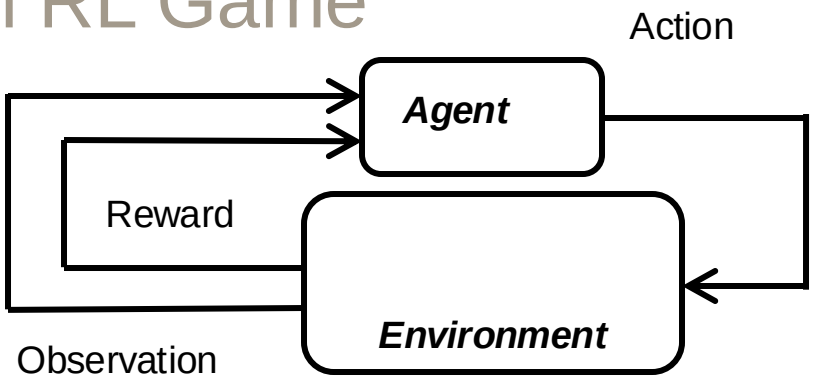
- Advantages of RL
  - learns a sequence of actions → cache-timing attack is a sequence
  - no dataset needed, just an env → a simulator/a real processor

Outline

- RL formulation of cache-timing attack exploration
- RL finds attacks on diverse configurations of caches
- RL finds attacks on real-hardware
- RL discovers new attack patterns

Cache-Timing Attack as an RL Game

- Agent: Attacker
- Environment: Cache
  - architecture simulator
  - cache in the processor
- Actions
  - attacker makes an access
  - attacker waits for victim access
  - attacker guesses the secret
- Observation
  - latency of attacker accesses

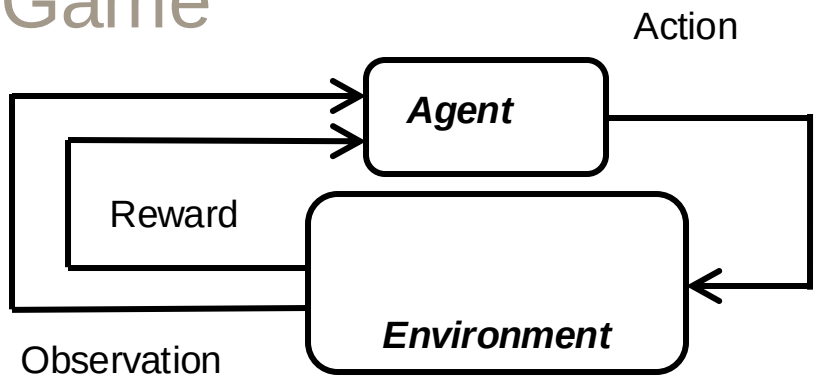


Summary of actions of existing attacks

Attacks	Attacker action	Victim action	Observations
prime+probe	access addrs	access an addr	attacker's latency
flush+reload	flush addrs	access an addr	attacker's latency
evict+reload	access addrs	access an addr	attacker's latency
evict+time	access addrs	access addrs	victim's latency

Cache-Timing Attack as a Game

- Reward
  - guess correct: positive reward
  - guess wrong: negative reward
  - each step: small negative reward
- Maximizing long-term reward
  - more correct guesses
  - fewer wrong guesses
  - fewer number of steps

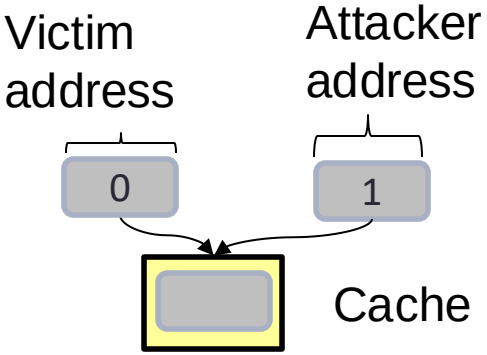


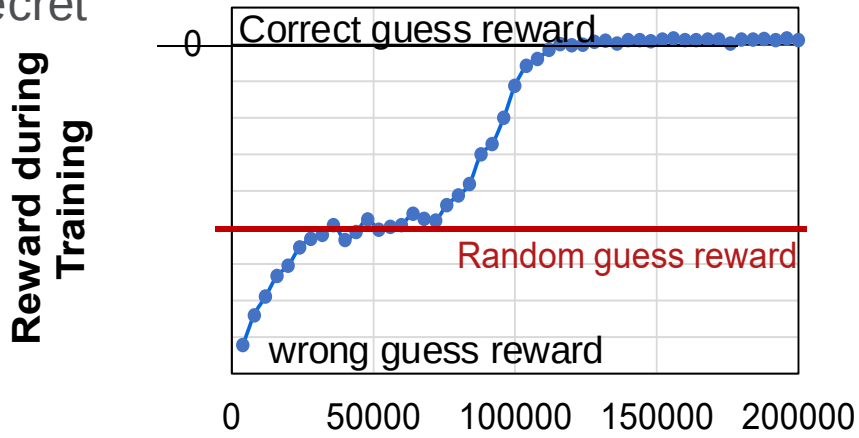
Summary of actions of existing attacks

Attacks	Attacker action	Victim action	Observations
prime+probe	access addrs	access an addr	attacker's latency
flush+reload	flush addrs	access an addr	attacker's latency
evict+reload	access addrs	access an addr	attacker's latency
evict+time	access addrs	access addrs	victim's latency

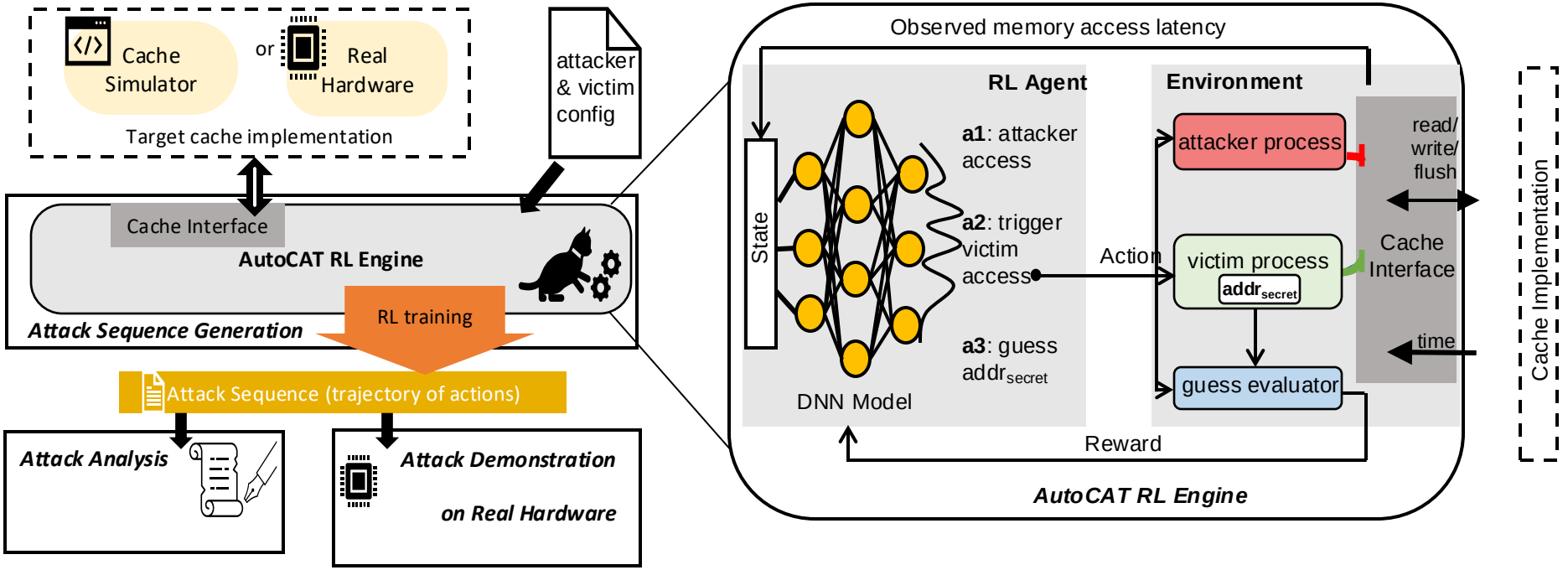
AutoCAT: A Simple Example

- Settings
  - 1 set 1-way cache
  - attacker can access address 1
  - victim secret: **access 0 (0) / no access (N)**
  - attacker want to infer whether victim secret **is 0/N**
- Attack found
  - step 1: attacker accesses 1
  - step 2: then wait for a while
  - step 3: attacker accesses 1 again
  - step 4: guess the secret **0/N**





AutoCAT: Framework Overview

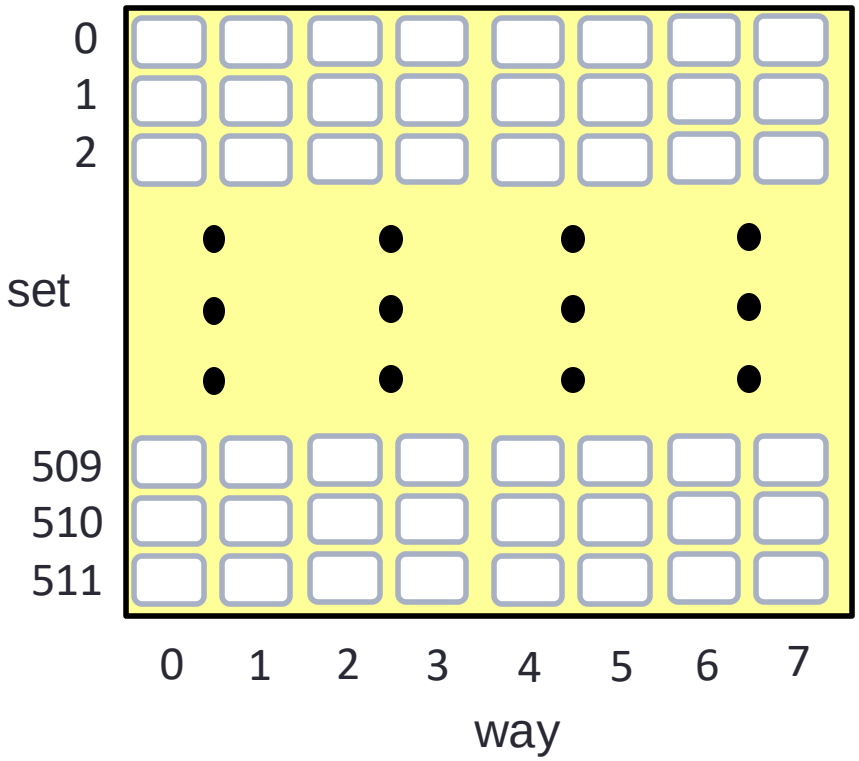


Outline

- RL formulation of cache-timing attack exploration
- RL finds attacks on diverse configurations of caches
- RL finds attacks on real-hardware
- RL discovers new attack patterns

AutoCAT: Attacks on Diverse Configurations

- Number of sets/ways
- Type of caches
  - direct-map/fully-associative/set-associative
- Replacement policies
  - least recently used (LRU)/re-reference interval prediction (RRIP)
- Prefetchers
  - none/stream/nextline
- Single-level/multi-level



AutoCAT: Attacks in Simulator

- Find attack patterns across 17 different configurations (No. 1-17)
  - including **direct-map, fully-associative, prefetchers, 2-level caches**

Excerpts from Table IV in the paper

No	Type	Ways	Sets	Victim address	Attacker address	Accuracy
1	Direct-map	1	4	0-3	4-7	100%
4	Fully-associative	4	1	0/E	4-7	100%
13	Fully-associative+nextline	8	1	0/E	0-15	100%
16	2-level	2	4	0-3	4-11	100%

Outline

- RL formulation of cache-timing attack exploration
- RL finds attacks on diverse configurations of caches
- RL finds attacks on real-hardware
- RL discovers new attack patterns

AutoCAT: Real Hardware

- AutoCAT finds attacks without knowing the replacement policy

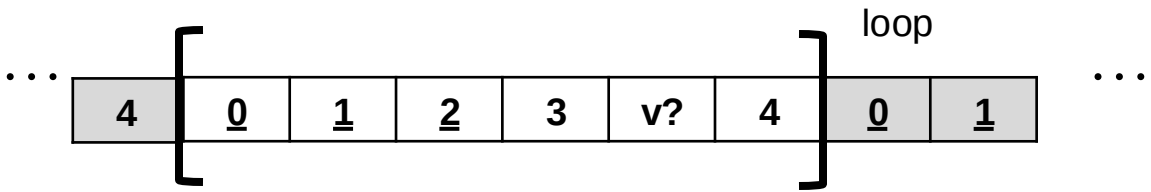
CPU	Level	Ways	Rep policy	Accuracy
Core i7-6700	L1	8	PLRU	100%
Core i7-6700	L2	4	Undocumented	99.9%
Core i7-6700	L3	4	Undocumented	100%
Core i7-7700K	L3	4	Undocumented	100%
Core i7-7700K	L3	8	Undocumented	99.3%
Core i7-9700	L1	8	PLRU	99.8%
Core i7-9700	L2	4	Undocumented	100%

Outline

- RL formulation of cache-timing attack exploration
- RL finds attacks on diverse configurations of caches
- RL finds attacks on real-hardware
- RL discovers new attack patterns

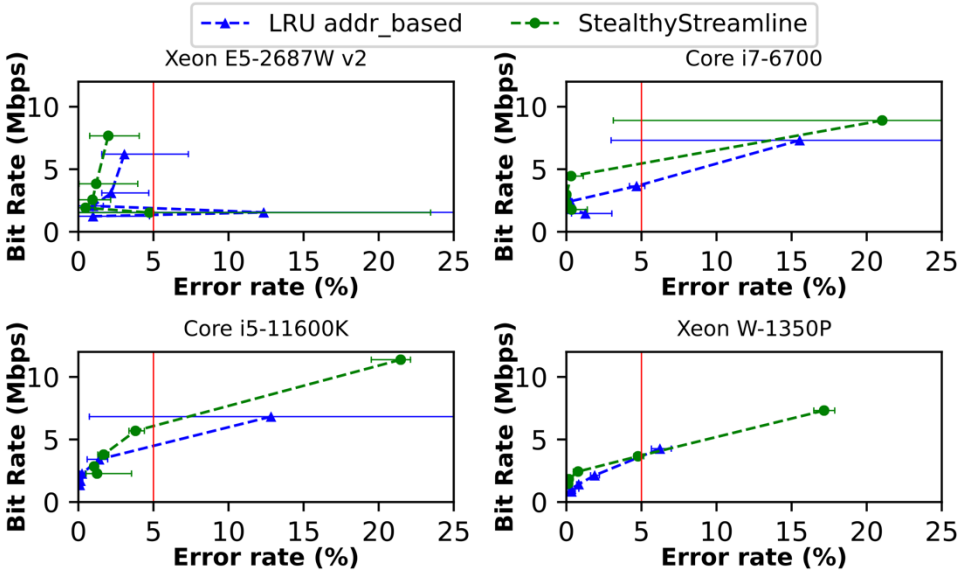
AutoCAT: A New Attack Pattern

- Setting:
  - 4-way cache
  - victim secret address from 0, 1, 2, 3
- Attack pattern:
  - attacker accesses 0, 1, 2, 3 first
  - victim accesses the secret address (always a hit)
  - attacker accesses 4, 0, 1, measure the timing of 0 and 1
  - depending on 0,1 hit/miss can infer the victim secret address



New Attack Pattern: StealthyStreamline

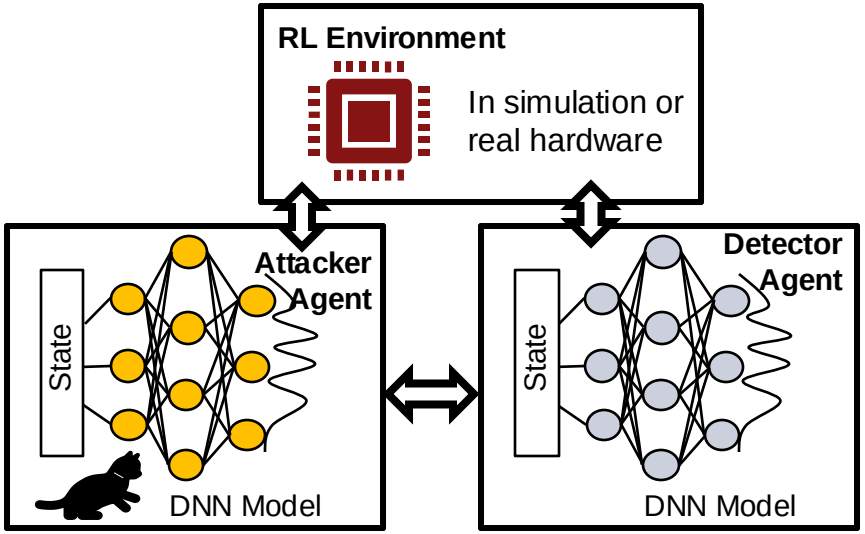
- No victim cache misses
- Works across different processors
  - 4 different Xeon/Core processor tested
- Higher bandwidth than the LRU-based attack



StealthyStreamline bandwidth and error rate (top-left corner is better)

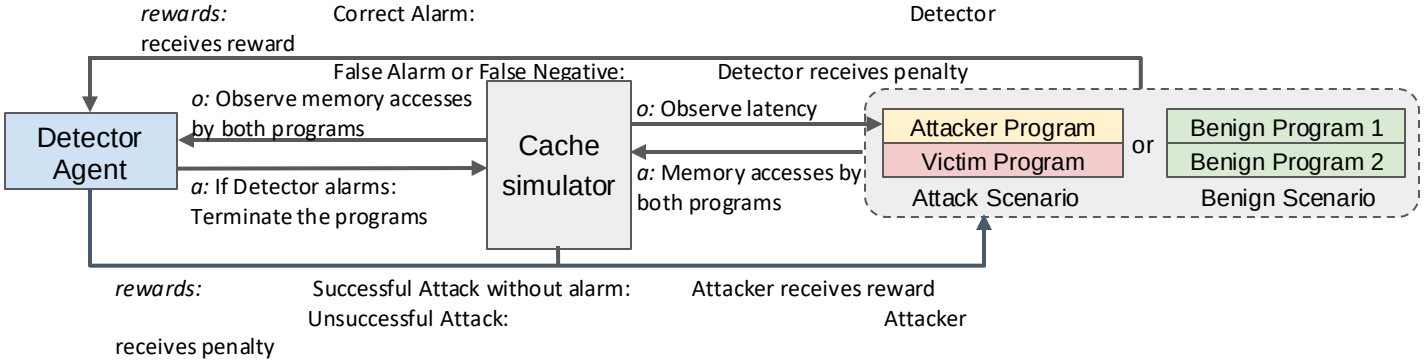
MACTA: A Multi-agent Reinforcement Learning Approach for Cache Timing Attacks and Detection

- Approach:
  - Multi-agent reinforcement learning (RL) for automatically exploring cache-timing attacks and detection schemes together.
- Key Findings:
  - Without any manual input from security experts,
    - the trained attacker is able to act more stealthily
    - the trained detector can generalize to unseen attacks
    - the trained detector is less exploitable to high-bandwidth attacks.



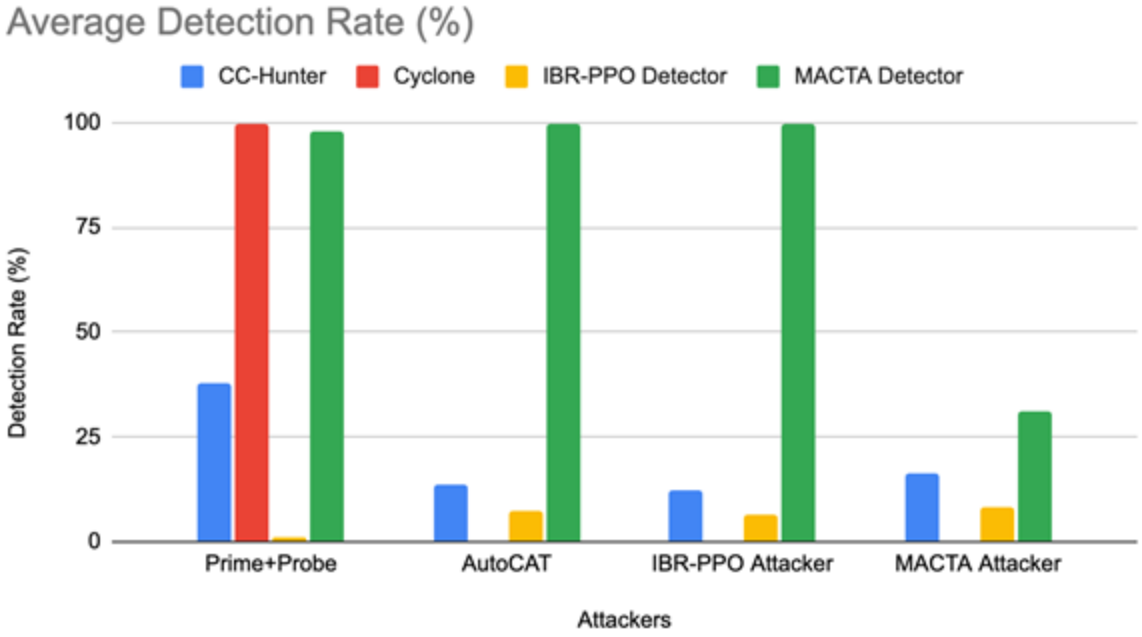
Multiagent RL Formulation

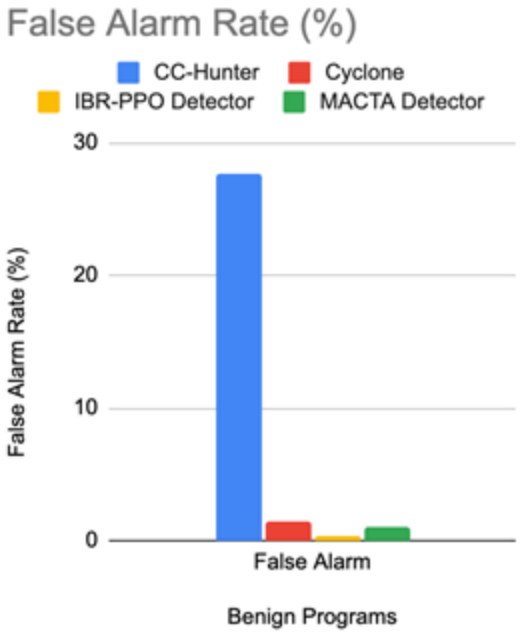
- To train a detector, we need both attacker scenario and benign Scenario.
- For each agent, there are observation, action, and rewards, respectively.



MACTA Results

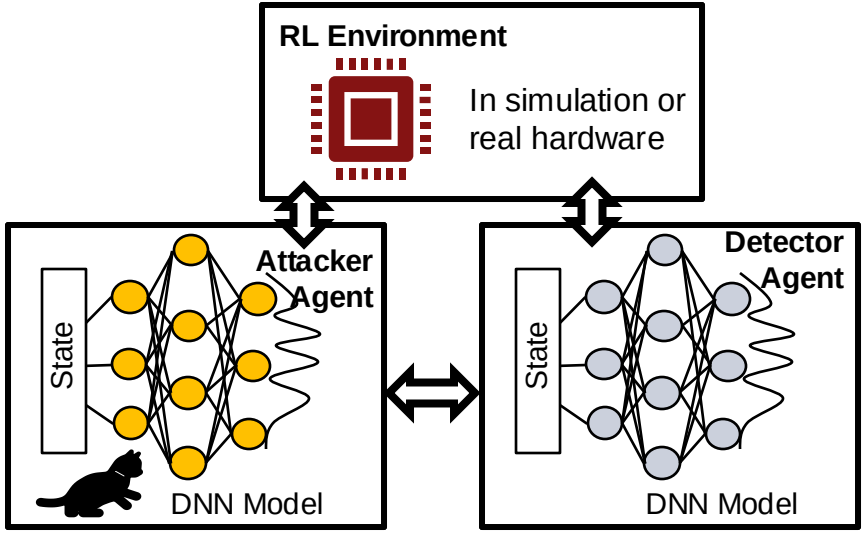
- Without any manual input from security experts,
  - the trained MACTA detector can generalize to unseen attacks





Conclusions and Future Works

- Reinforcement learning shows promising results for exploring the attacks and detection HW vulnerabilities for CPS
- Would this method scale?
- How to understand the results of the model?
- How about other security problems?



Learn more at https://rl4cas.github.io

Acknowledgements

Thank you for your time!








CORNELL UNIVERSITY FOUNDED A.D. 1865

Meta

VIRGINIA TECH

More in the Paper

- Bypassing defense and detection techniques
  - partition-locked (PL) cache
  - autocorrelation-based detection similar to CC-Hunter [MICRO14]
  - ML-based detection similar to Cyclone [MICRO19]

- Discussions
  - comparison with search algorithms
    - less number of steps compared with exhaustive search
  - future extensions
    - automated attack analysis
    - scalability of the RL model

Conclusion

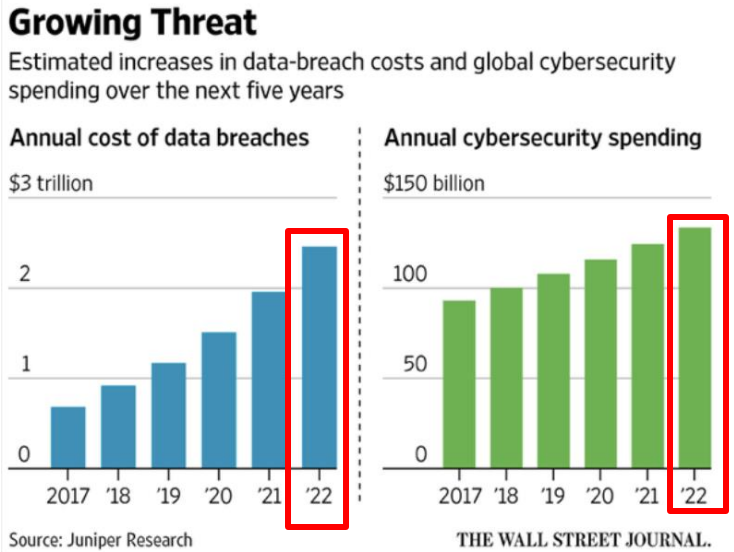
- AutoCAT uses RL to explore cache-timing attacks in processors
  - without explicit specification of processor design
  - without knowing existing attack patterns
- AutoCAT found attack patterns
  - on many configurations in the cache simulator/real hardware
  - a new attack pattern: StealthyStreamline

Artifact available at: https://github.com/facebookresearch/AutoCAT

Introduction: Cybersecurity

- Vulnerability is everywhere
  - Stuxnet: nuclear power plant
  - Cambridge Analytica: social networks
  - Log4j: web infrastructure
  - Spectre/Meltdown: computer hardware

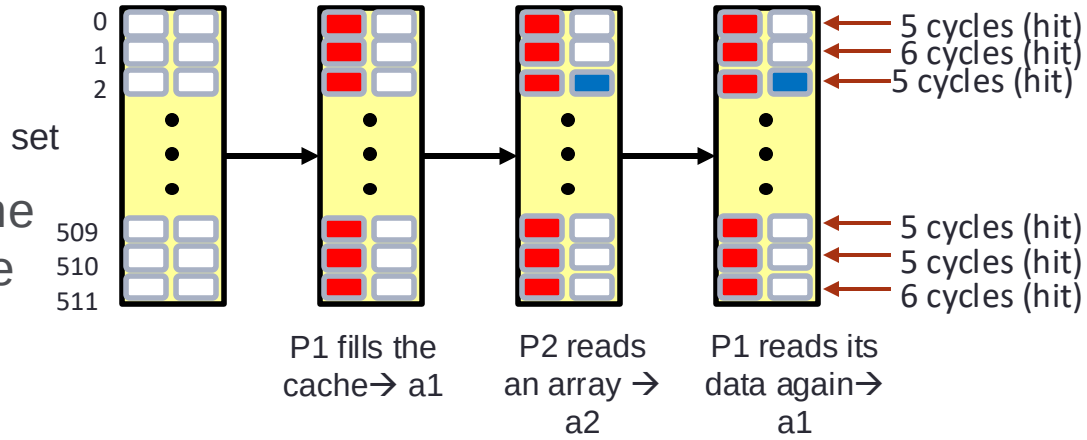
- Huge dollars spent
  - 2.5 trillion USD = GDP of UK (5th largest country)
  - finding and resolving vulnerabilities



Source: fair institute

Cache Defense Mechanism

- Partition
  - Attacker process and victim process uses their own cache lines without sharing with the others
  - E.g., PLCache
    - Example: 4-way PLCache
      - Way 0: victim process exclusive
      - Way 1-3: attacker process exclusive
    - Can AutoCAT find attacks on PLCache?



AutoCAT's Attack on PLCache

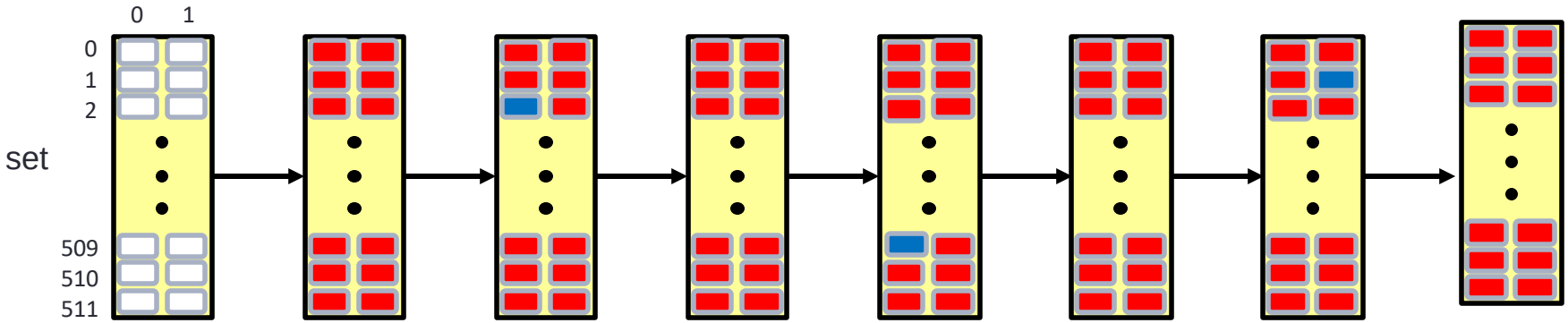
Next action	Victim access 0 when triggered		Victim no access when triggered	
access 1	0 1	hit	0 1	hit
Trigger victim	0 1	N.A.	0 1	N.A.
access 3	0 1	miss	0 1	miss
access 3	0 1 3	hit	0 1 3	hit
access 2	0 1 3	miss	0 1 3	miss
access 5	0 1 3 2	miss	0 1 3 2	miss
access 5	0 5 3 2	**hit**	0 1 3 2	**miss**

LRU addr 1 not locked, can be replaced

LRU addr 0 locked, cannot be replaced

Cache Timing Attack Detection

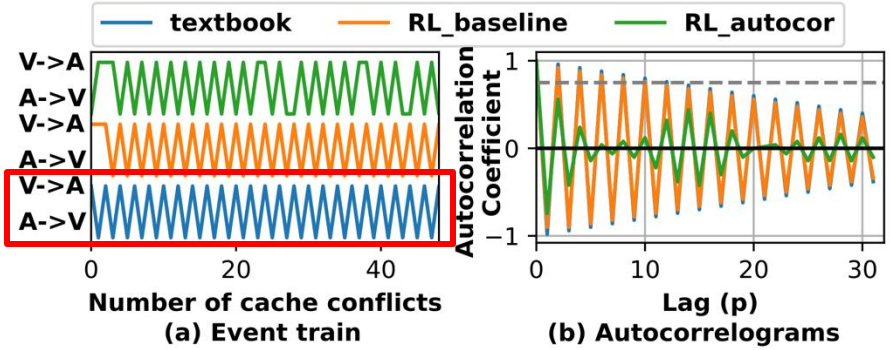
- The cache access pattern by the attacker have specific characteristics



Steps	P1 fills the cache	P2 reads an array	P1 reads the cache	P2 reads an array	P1 reads the cache	P2 reads an array	P1 reads the cache
Replacement events	none	replace	replace	replace	replace	replace	replace
Event encoding		0	1	0	1	0	1

AutoCAT can Bypass the Detection

- AutoCAT can generate attacks that bypass CC-Hunter
  - Textbook: prime+probe
  - RL_baseline: training without considering the CCHunter
  - RL_autocor: training with consider high autocor as penalty



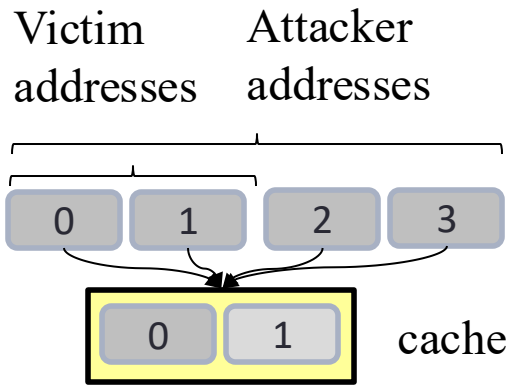
- AutoCAT can generate attacks that evade SVM detection
  - RL_SVM: training with considering SVM detection penalty

TABLE IX: Comparison of bit rate, guess accuracy and detection rate by the SVM.

sampled attacker	bit rate (guess/step)	attack accuracy	SVM detection rate
textbook	0.1625	1	1
RL_baseline	0.228	0.990	0.907
RL_SVM	0.150	0.964	0.021

AutoCAT: A Simple Example

- Settings
  - 1 set 2 way cache LRU policy
  - Attacker can access address 0, 1, 2, 3
  - Victim can access 0 or 1
  - Attacker want to infer whether victim accesses 0 or 1
- Reward
  - Correct guess: 200
  - Wrong guess: -10,000



Next action	Victim access 0 when triggered		Victim access 1 when triggered	
Trigger victim	0 1	hit	0 1	hit
access 3	0 1	N.A.	0 1	N.A.
access 1	0 3	**miss**	3 1	**hit**

Indicates the next block to be evicted

Attacker can infer $addr_{secret}$ by checking **hit** or **miss**

Covert and Side Channels

- Computation affects the physical world, and an observer can measure physical effects
- Confidential information may leak through physical properties not intended for communications
  - Timing, power consumption, EM, temperature, acoustic, etc.
- Covert channels
  - Use unintended physical properties to transmit information without the authorization or knowledge of a system
- Side channels
  - Unintentional covert channels

Timing Channel Protection

- Completely remove timing dependence
  - Secure, but expensive
- More empirical protection
  - Reduce the timing dependence (noise, coarser-grained resource allocation, etc)
  - Detect an attack
  - Less expensive, but difficult to provide a security guarantee
- Use ML to automatically generate attacks?
  - A game between an attacker and a defender

Side-Channel Attack: An Example

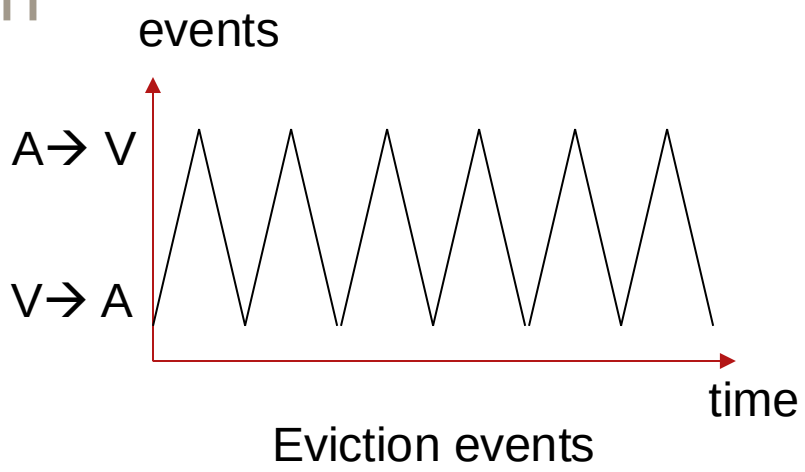
- Infer secret information from target device by observing power consumption
- Threat models require physical access or proximity to device

Source: Cryptography Research

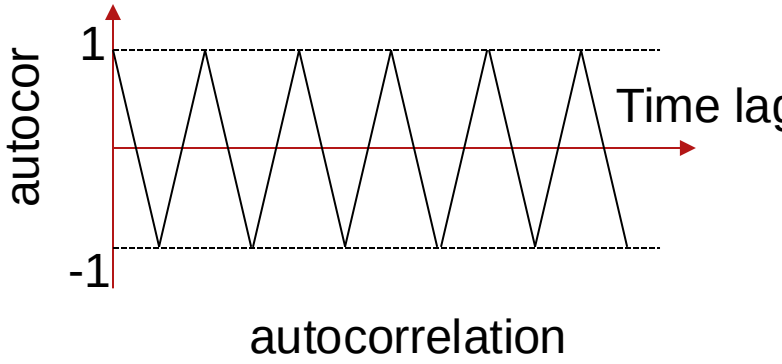
Source: Wikipedia

Cache Timing Attack Detection

- CC-Hunter: calculate the autocorrelation of eviction encode
  - High autocorrelation → likely an attack
  - Low autocorrelation → likely a benign program

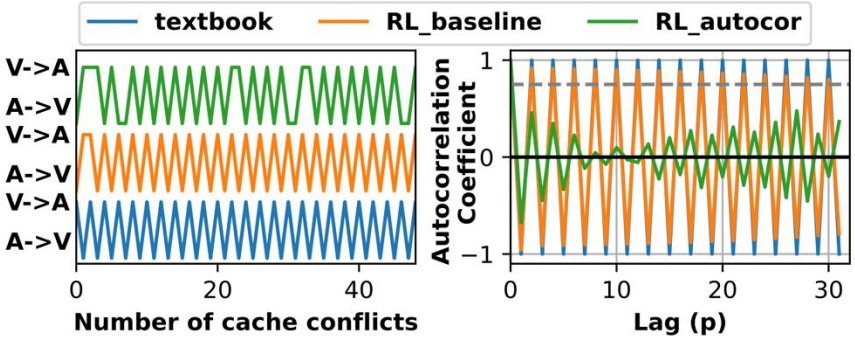


- Cyclone
  - Use SVM classifier to detect an attack



AutoCAT can Bypass the Detection

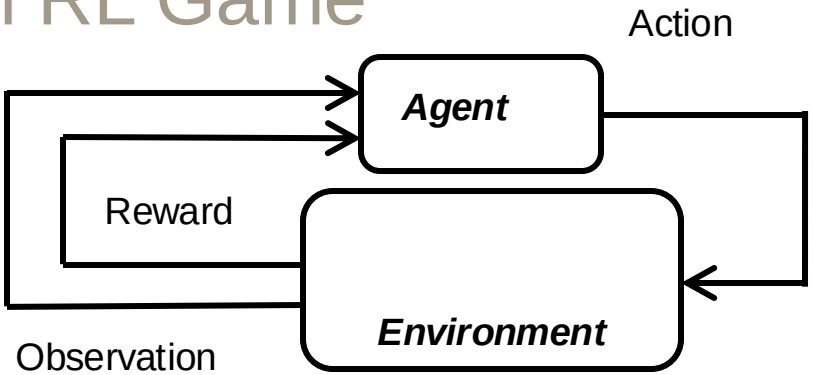
- AutoCAT can generate attacks that bypass CC-Hunter
  - Textbook: prime+probe
  - RL_baseline: training without considering the CCHunter
  - RL_autocor: training with consider high autocor as penalty
- AutoCAT can generate attacks that evade SVM detection
  - RL_SVM: training with considering SVM detection penalty

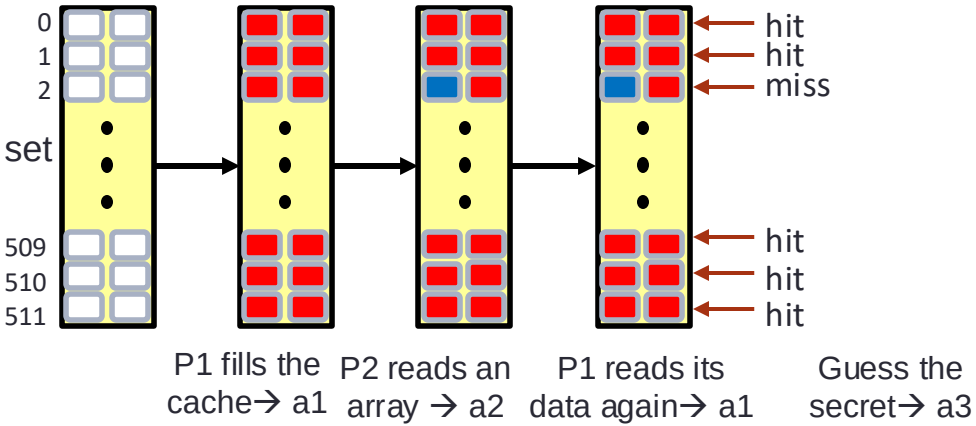


attacker	Bit rate	Accuracy	Detection rate
textbook	0.1625	1.0	0.973
RL_baseline	0.229	0.989	0.933
RL_SVM	0.216	0.997	0.519

Cache-Timing Attack as an RL Game

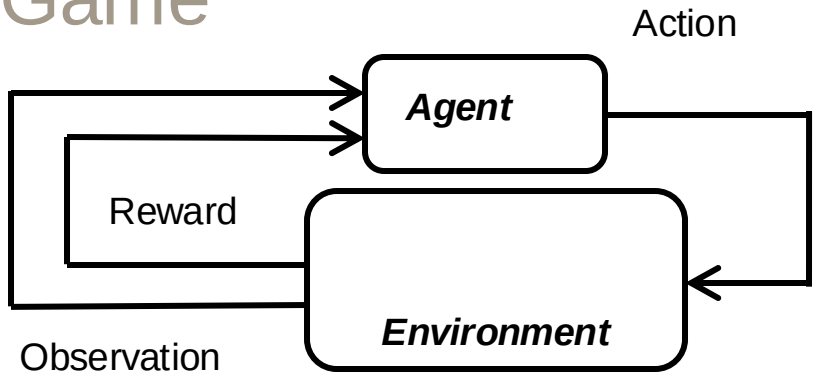
- Agent: Attacker
- Environment: Cache
  - architecture simulator
  - cache in the processor
- Actions
  - a1: attack makes an access
  - a2: wait for victim access
  - a3: guess the secret
- Observation
  - latency of attacker access

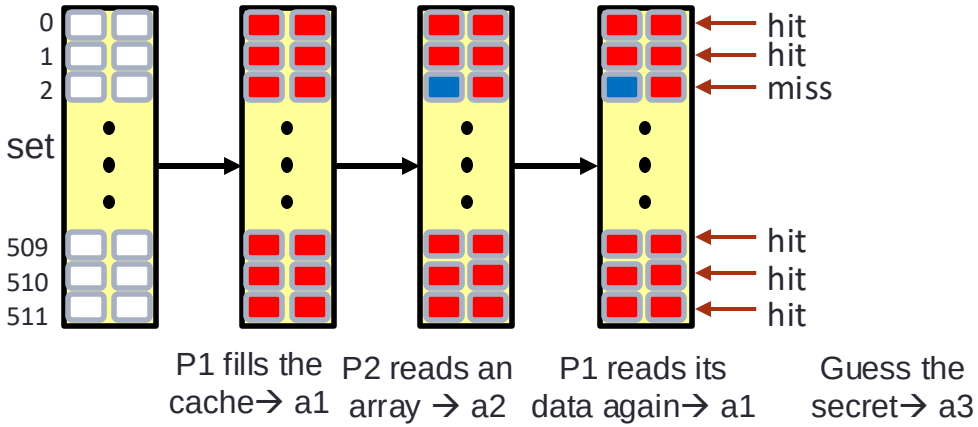




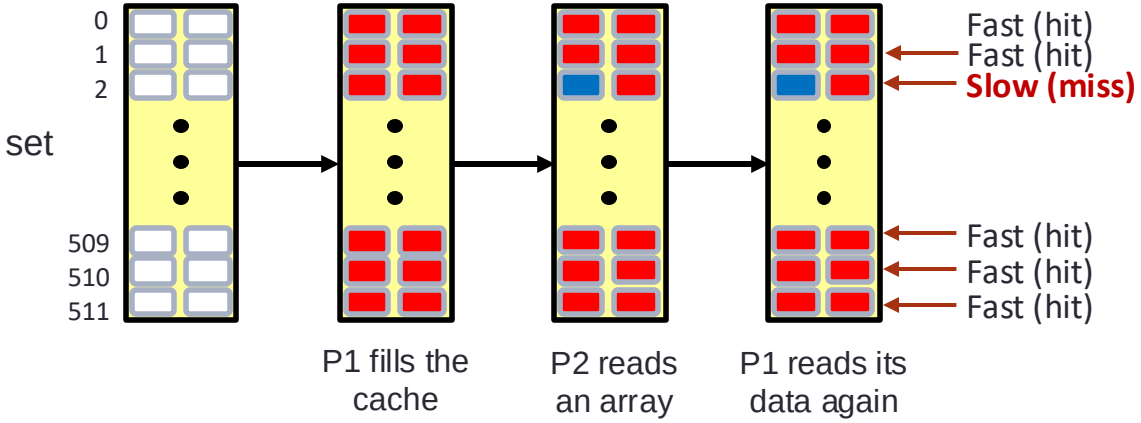
Cache-Timing Attack as a Game

- Reward
  - guess correct: positive reward
  - guess wrong: negative reward
  - each step: small negative reward
- Maximizing long-term reward
  - More correct guess
  - Less wrong guess
  - Less number of steps → shorter attack sequence



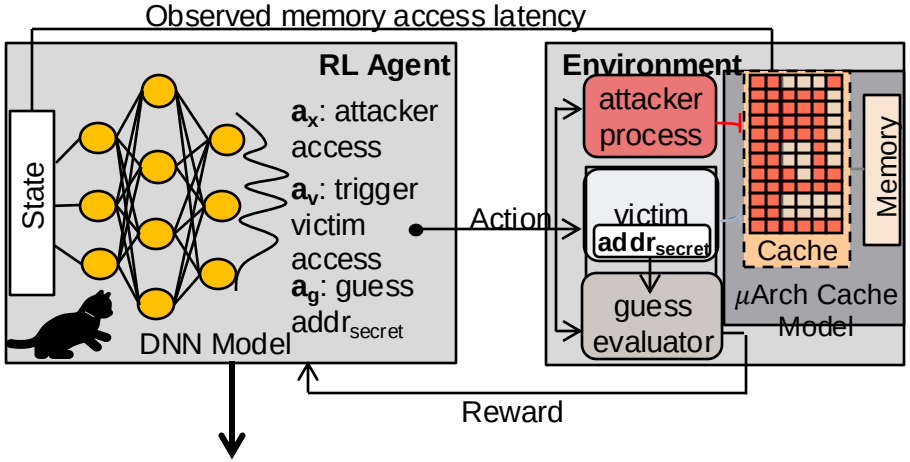


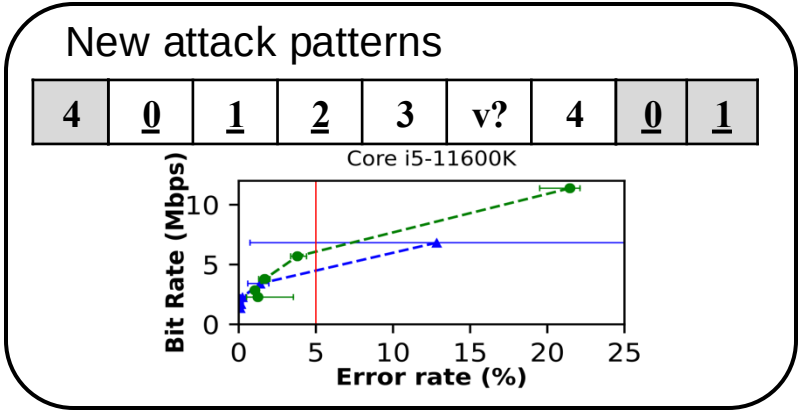
Cache Timing Attack



Executive Summary

- We use RL to automatically explore cache timing attacks in processors
- We find attacks on a diverse configurations of caches
  - Different replacement policies
  - Different defense mechanisms
- We discover StealthyStreamline attacks
  - Bypass performance counter-based detection
  - Higher bandwidth





Prior Work on ML for Side Channel

- Focuses on using ML to analyze existing power side-channel traces and predict secrets
  - Use a supervised learning model to analyze traces
  - Auto-encoder to learn representations