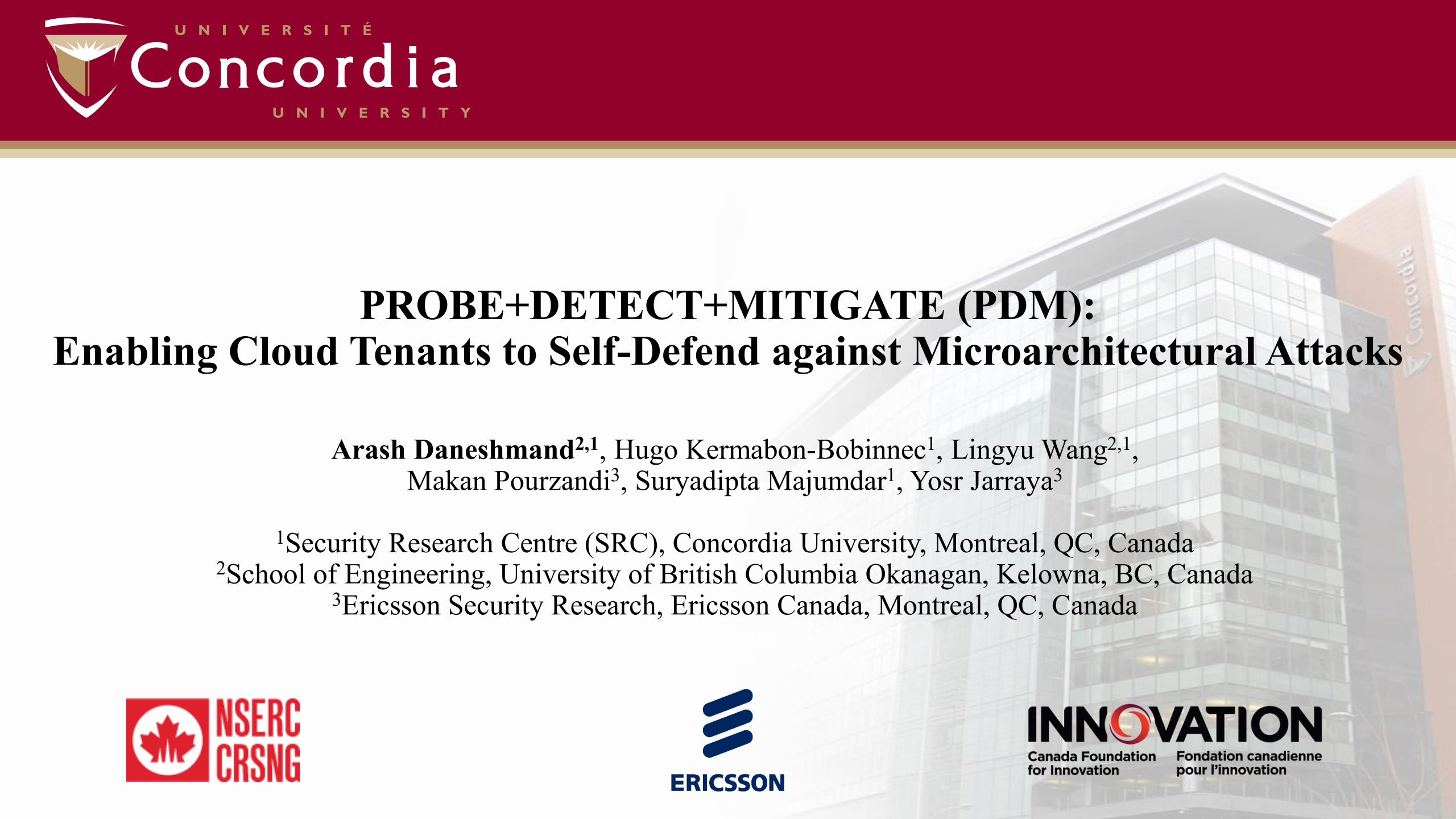


PROBE+DETECT+MITIGATE (PDM): Enabling Cloud Tenants to Self-Defend against Microarchitectural Attacks

Arash Daneshmand[2,1], Hugo Kermabon-Bobinnec[1], Lingyu Wang[2,1],
Makan Pourzandi[3], Suryadipta Majumdar[1], Yosr Jarraya[3]

[1]Security Research Centre (SRC), Concordia University, Montreal, QC, Canada
[2]School of Engineering, University of British Columbia Okanagan, Kelowna, BC, Canada
[3]Ericsson Security Research, Ericsson Canada, Montreal, QC, Canada

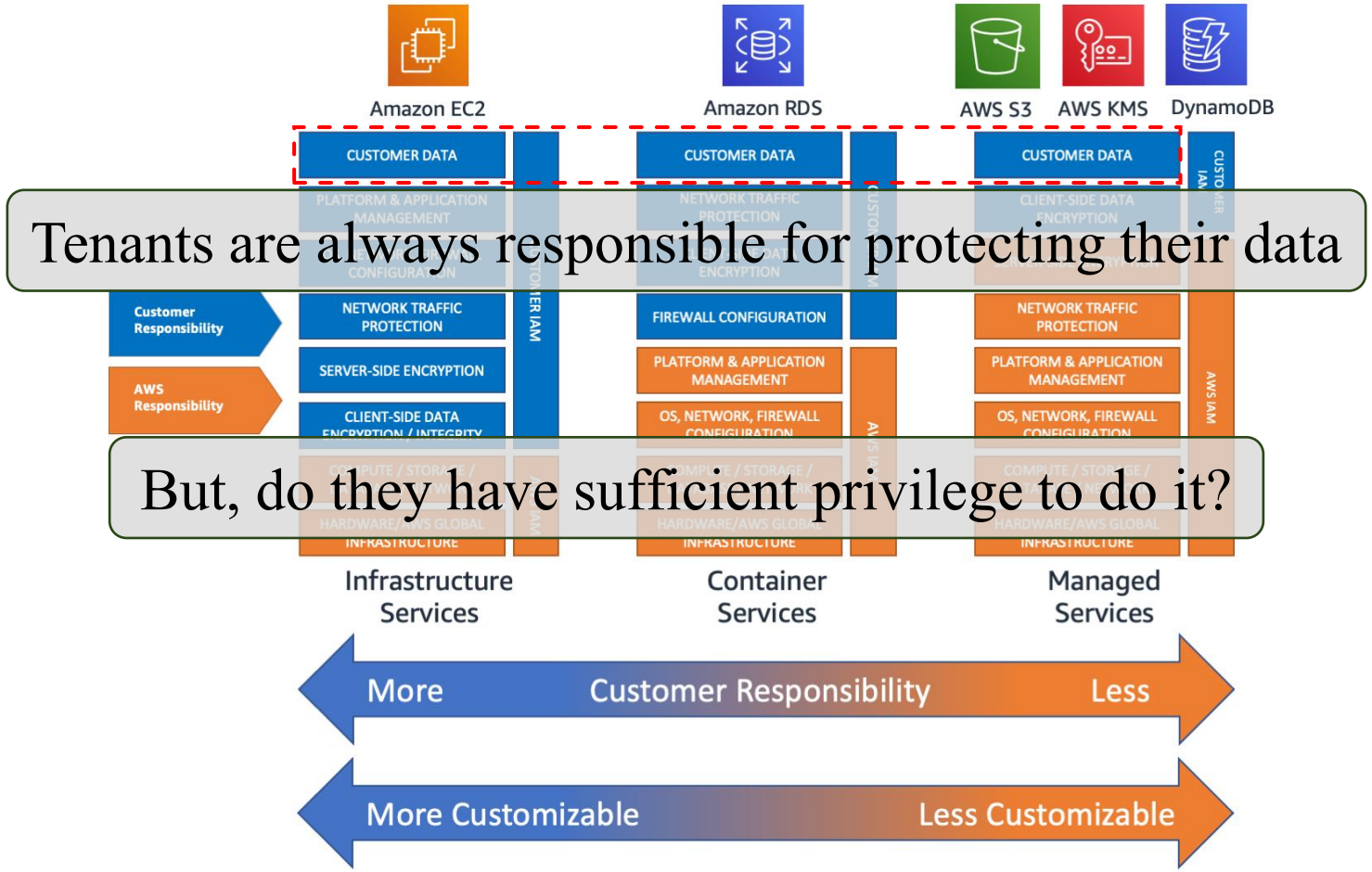
Outline

- Introduction
- Motivation
- Methodology
- Evaluation
- Conclusion

Outline

- Introduction
- Motivation
- Methodology
- Evaluation
- Conclusion

Introduction – Shared Responsibility Model



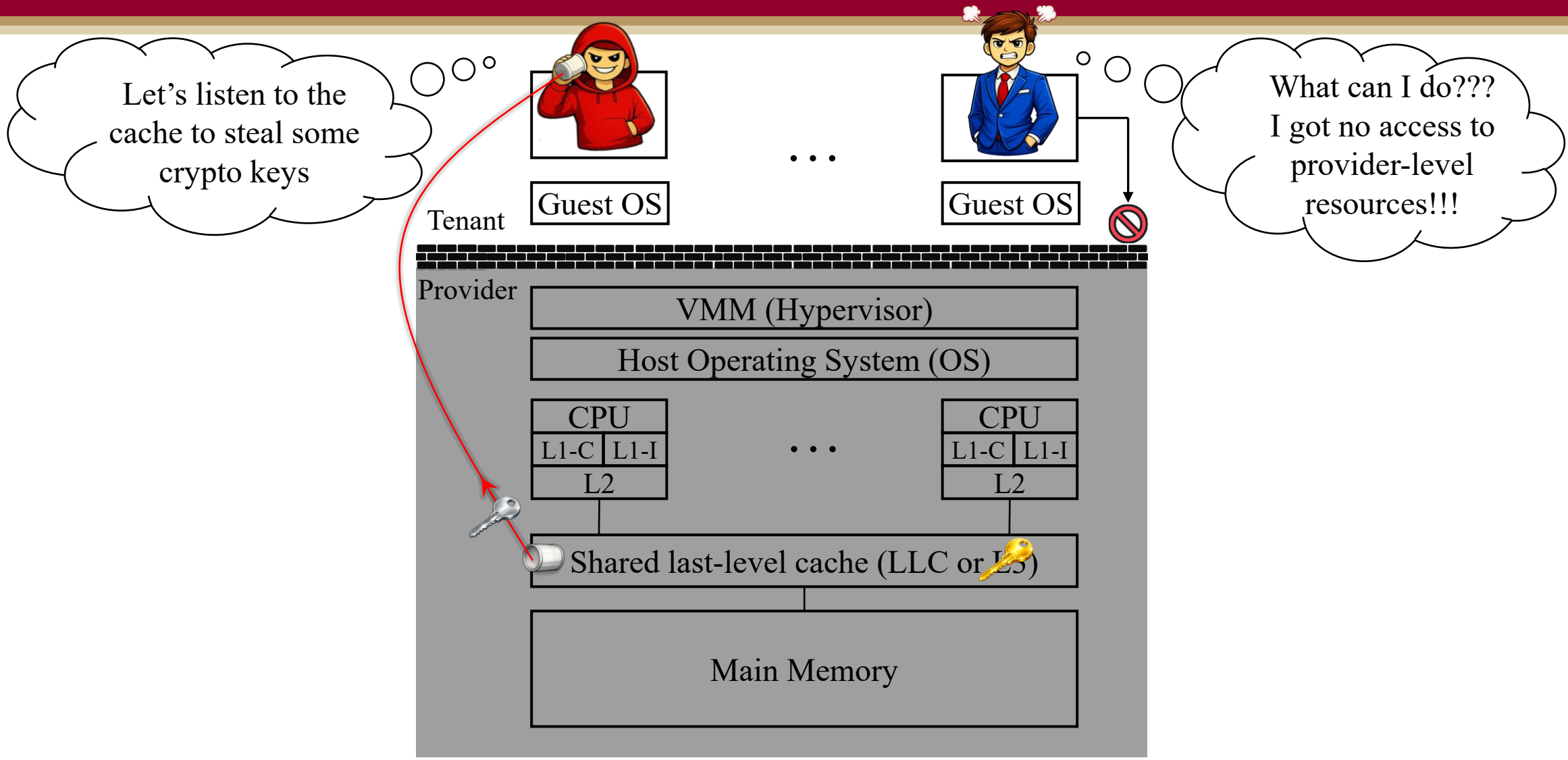
Tenants are always responsible for protecting their data

But, do they have sufficient privilege to do it?

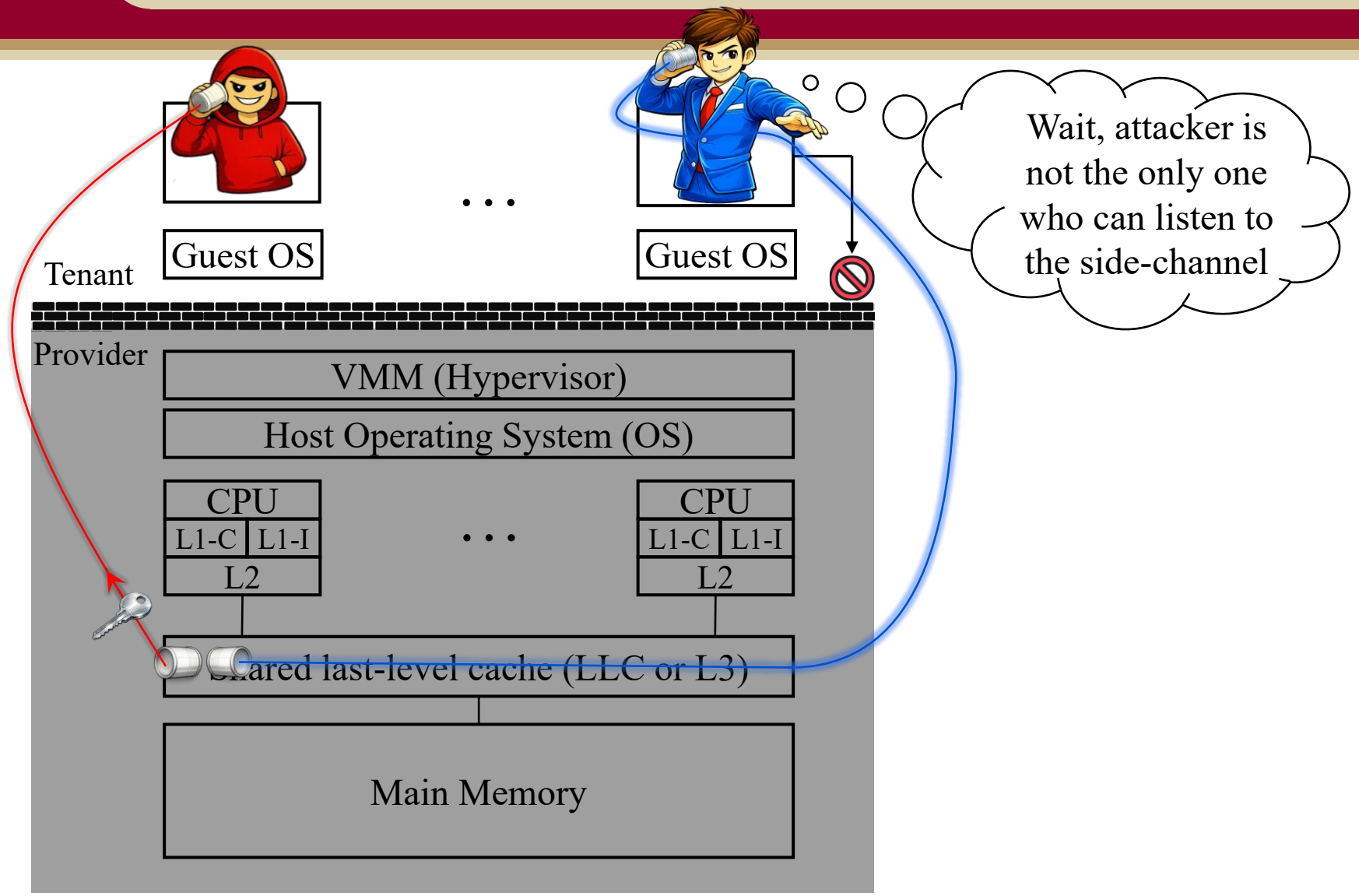
AWS Shared Responsibility Model [1]

[1]: https://aws.amazon.com/blogs/industries/applying-the-aws-shared-responsibility-model-to-your-gxp-solution/

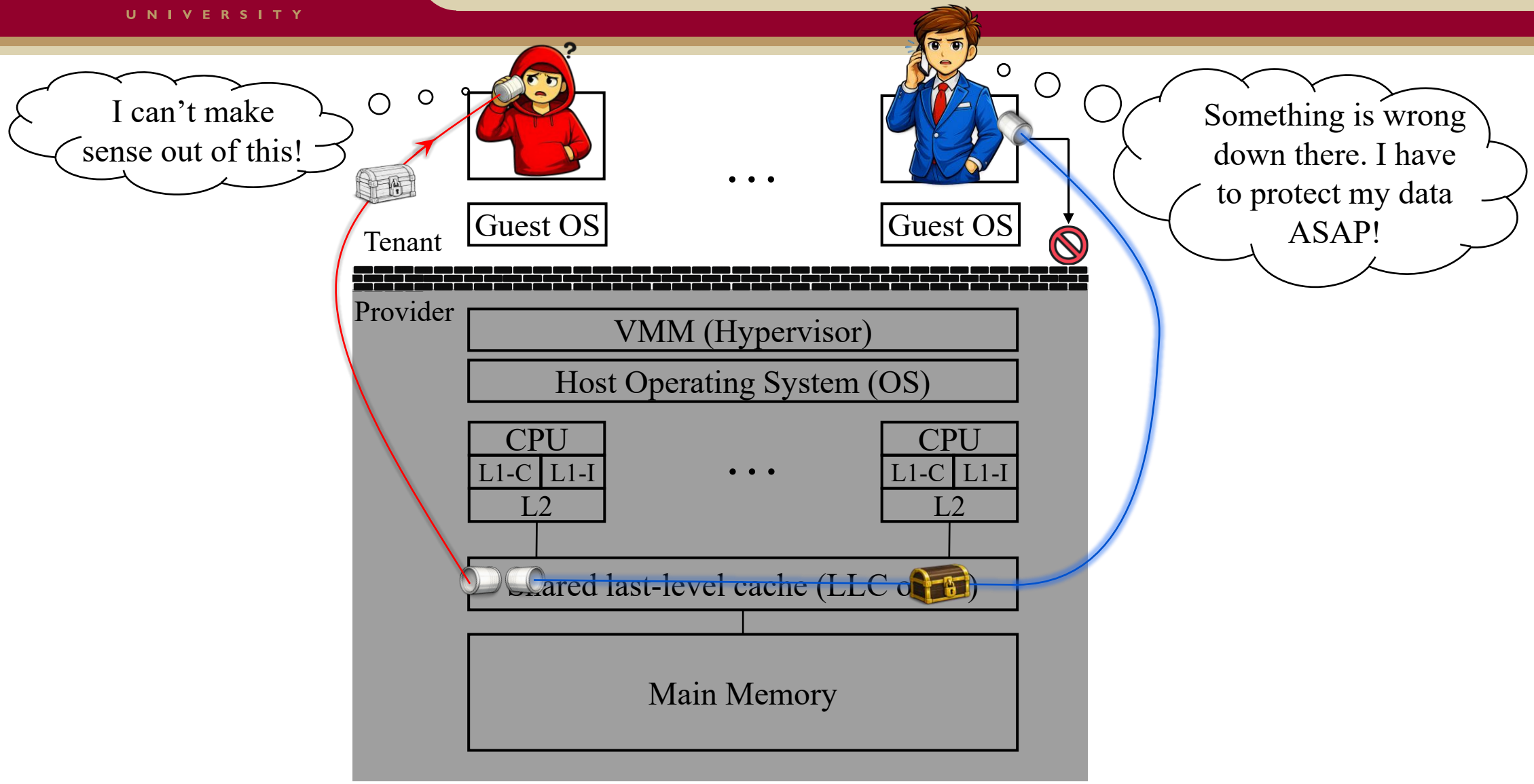
Motivation


Let's listen to the cache to steal some crypto keys
Guest OS
Guest OS
What can I do??? I got no access to provider-level resources!!!
Tenant
Provider
VMM (Hypervisor)
Host Operating System (OS)
CPU
L1-C
L1-I
L2
CPU
L1-C
L1-I
L2
Shared last-level cache (LLC or L3)
Main Memory

Detection


Wait, attacker is not the only one who can listen to the side-channel
Guest OS
Guest OS
Tenant
Provider
VMM (Hypervisor)
Host Operating System (OS)
CPU
L1-C
L1-I
L2
CPU
L1-C
L1-I
L2
Shared last-level cache (LLC or L3)
Main Memory

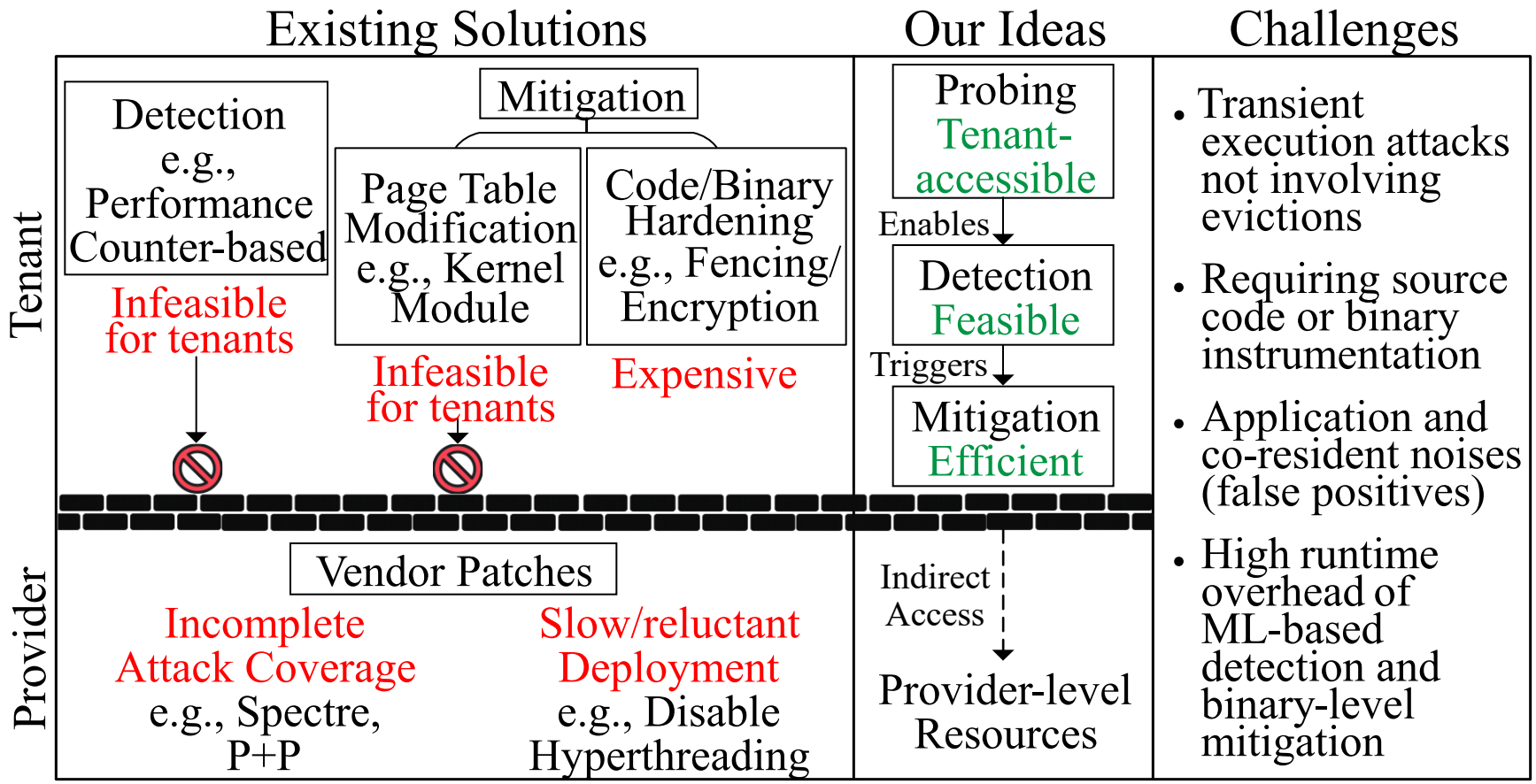
Mitigation


I can't make sense out of this!
Something is wrong down there. I have to protect my data ASAP!
Tenant
Guest OS
Guest OS
Provider
VMM (Hypervisor)
Host Operating System (OS)
CPU
L1-C
L1-I
L2
CPU
L1-C
L1-I
L2
Main Memory

Outline

- Introduction
- Motivation
- Methodology
- Evaluation
- Conclusion

Motivation

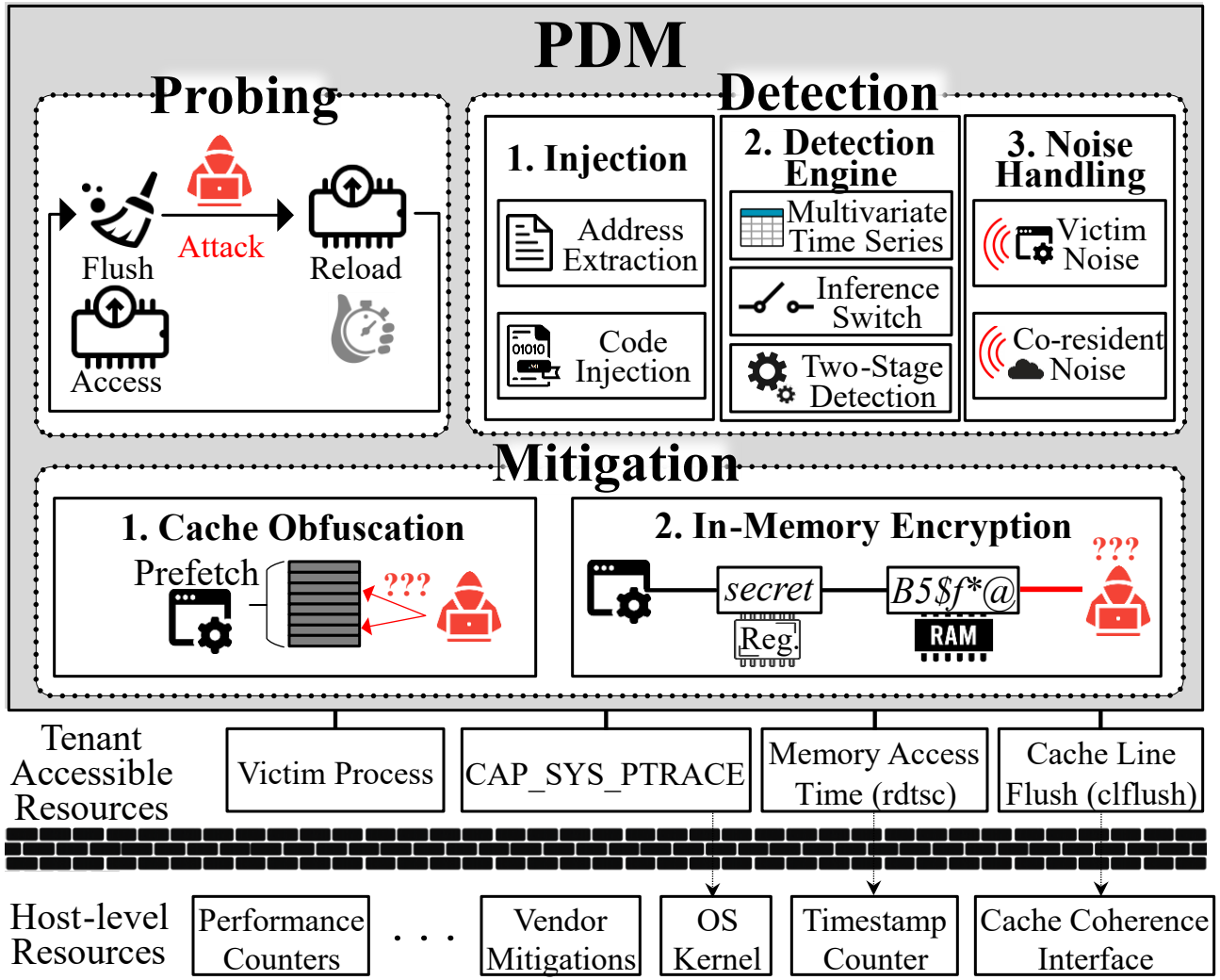

Existing Solutions
Our Ideas
Challenges
Tenant
Provider
Detection
e.g., Performance Counter-based
Infeasible for tenants
Mitigation
Page Table Modification e.g., Kernel Module
Infeasible for tenants
Code/Binary Hardening e.g., Fencing/ Encryption
Expensive
Vendor Patches
Incomplete Attack Coverage
e.g., Spectre, P+P
Slow/reluctant Deployment
e.g., Disable Hyperthreading
Probing Tenant-accessible
Enables
Detection Feasible
Triggers
Mitigation Efficient
Indirect Access
Provider-level Resources
• Transient execution attacks not involving evictions
• Requiring source code or binary instrumentation
• Application and co-resident noises (false positives)
• High runtime overhead of ML-based detection and binary-level mitigation

Outline

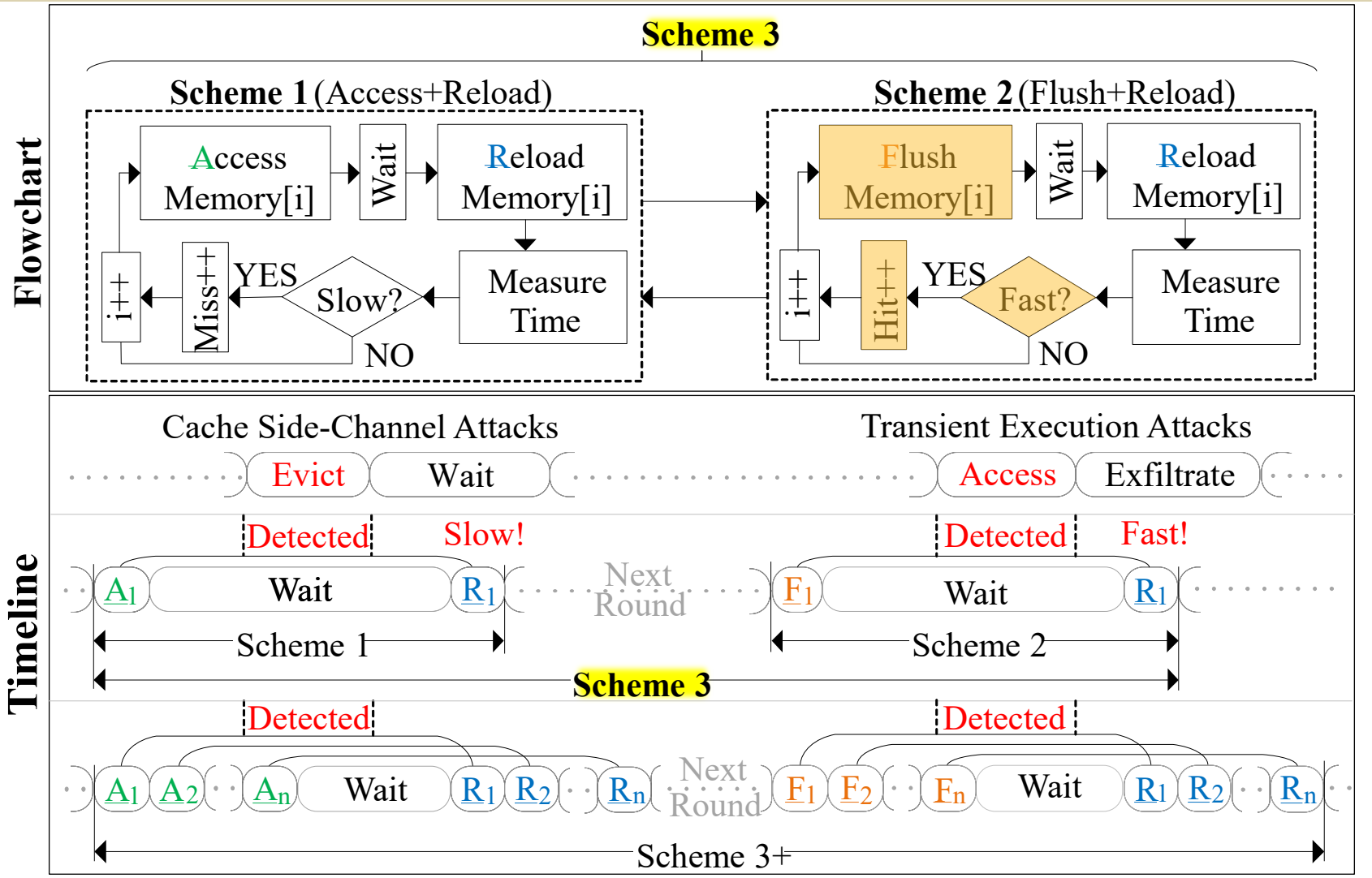
- Introduction
- Limitations
- Methodology
- Evaluation
- Conclusion

Overview

- Probing
- Detection
  - Injection
    - Address Extraction
    - Code Injection
  - Detection Engine
    - Multivariate Time Series
    - Inference Switch
    - Two-stage Detection
  - Noise Handling
    - Victim Noise
    - Co-resident Noise
- Mitigation
  - Cache Obfuscation
  - In-memory Encryption

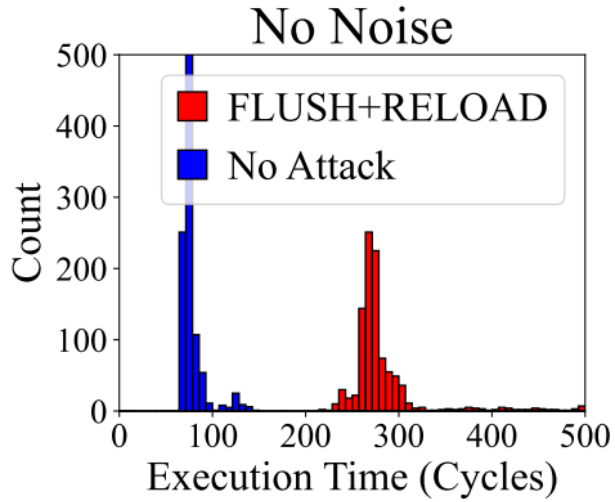


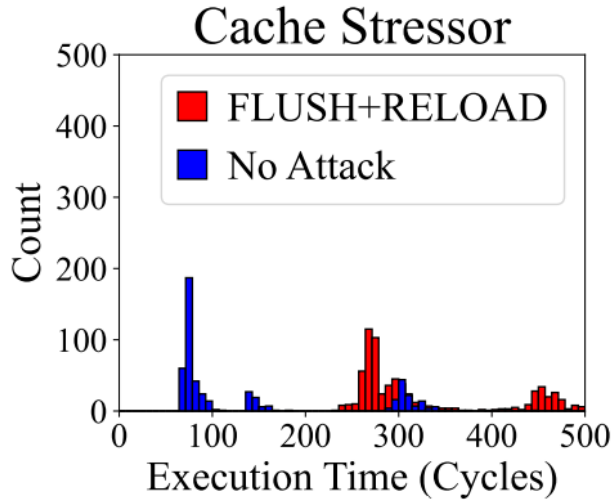
I. Probing – Schemes and Timeline

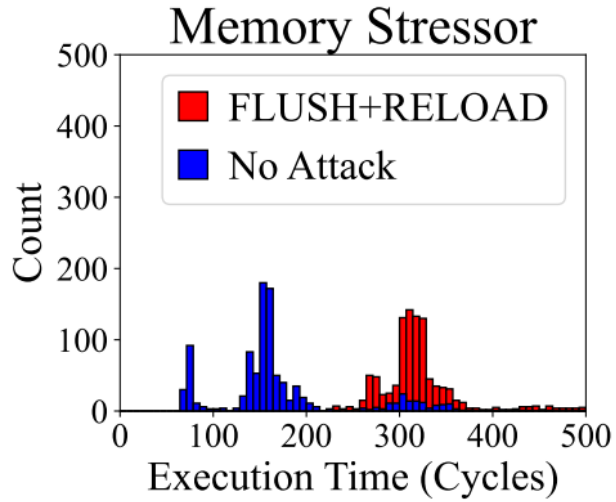


I. Probing – Examples of Collected Data

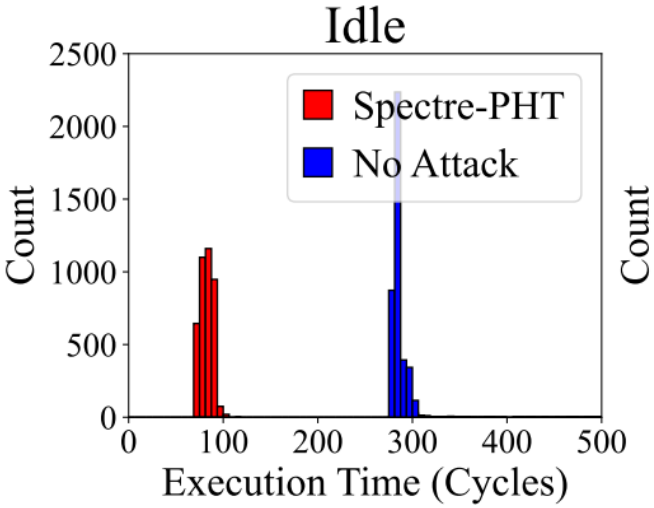
Under Different Levels of Background Noise

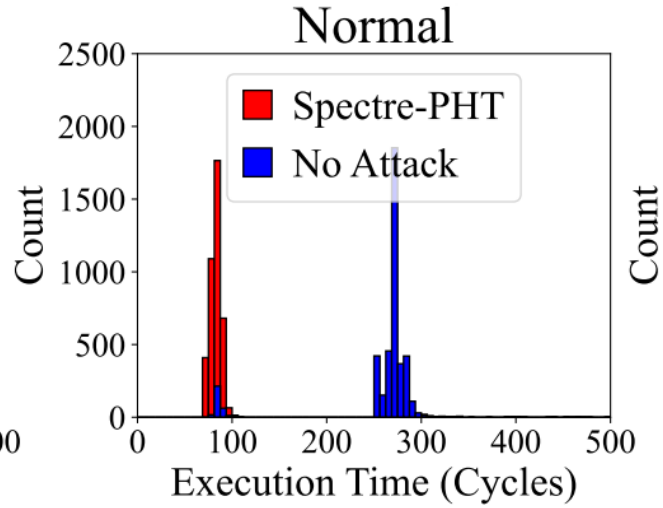


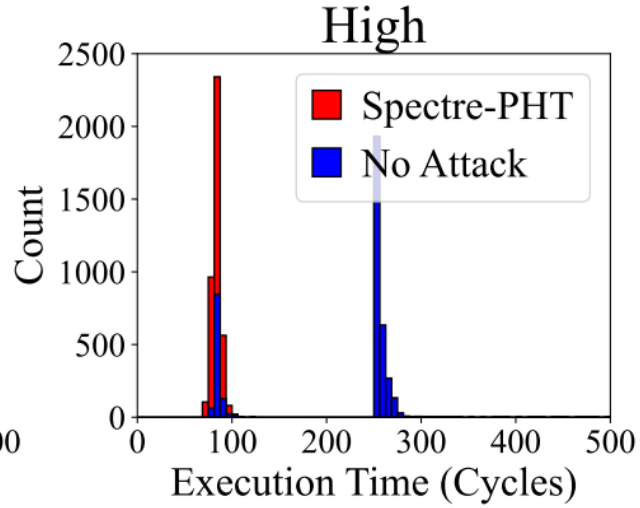




Under Different Levels of Application Workload

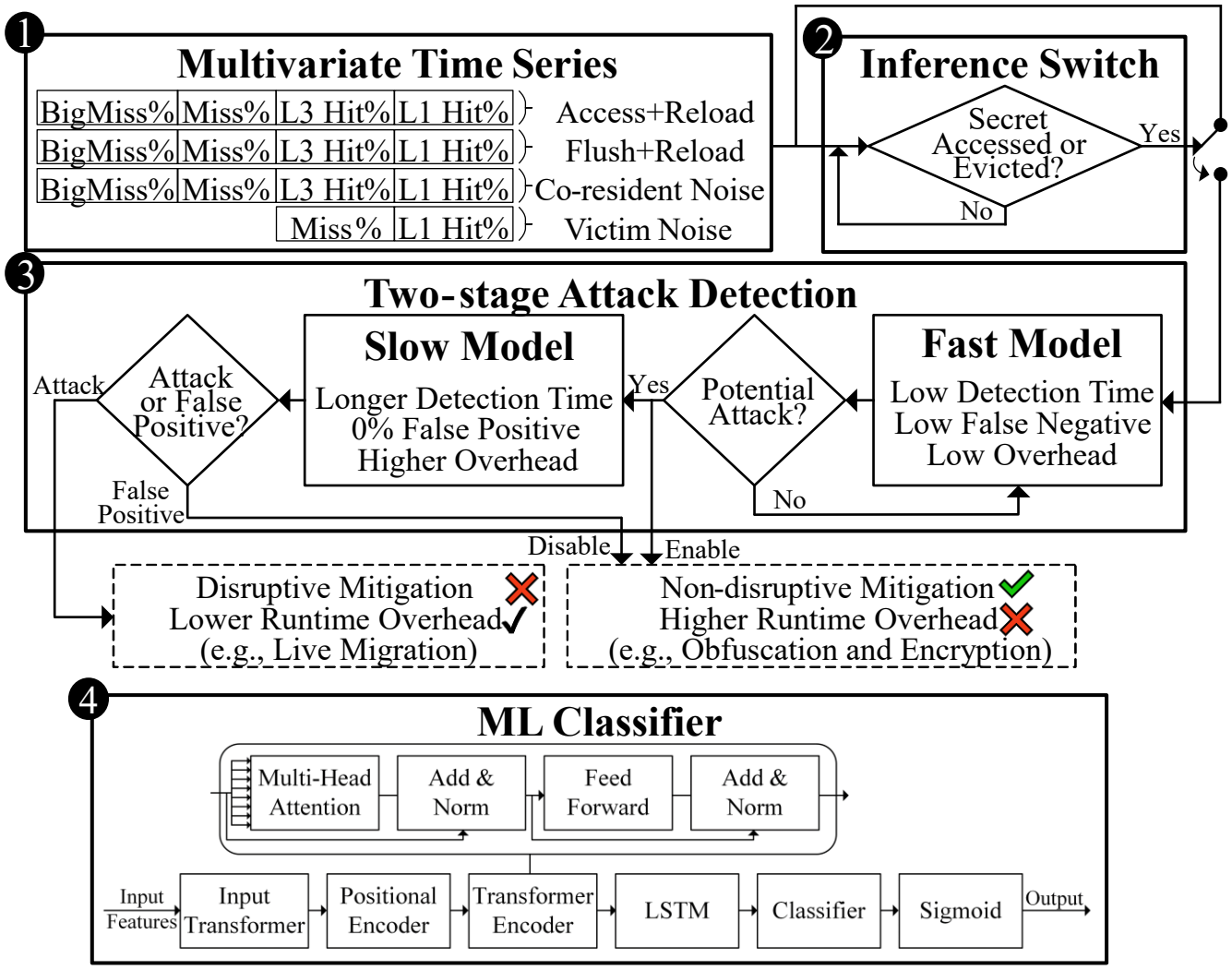






II. Detection

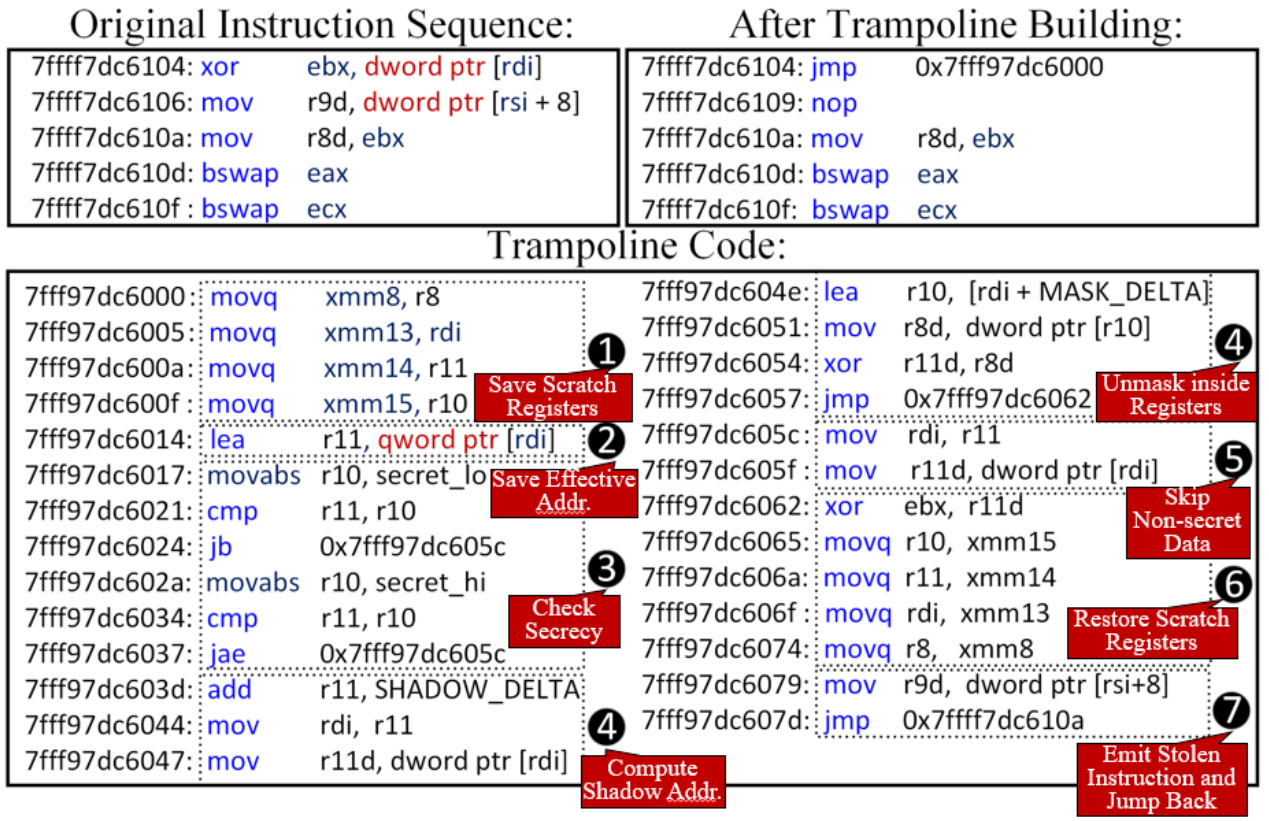
- Multivariate time series
  - Enables the analysis of dependencies between all the features
  - Ratio of probes that hit each level of the cache-memory hierarchy, separated with different thresholds
- Inference switch
  - Constant ML inference introduces unnecessary overhead and delay
  - Only trigger ML if the secrets were accessed or evicted
- Two-stage attack detection
  - Balance between the detection lead time and accuracy
  - Fast classifier triggers non-disruptive mitigation while slow classifier confirms the decision
- ML classifier
  - Hybrid deep learning model combining transformer and bi-directional LSTM
  - Runtime inference using ONNX



III. Mitigation

- Mitigation for Cache Side-Channel Attacks
  - Deploying cache obfuscation
  - Prefetching many pages into the cache
  - Increase the noise observed by attackers

- Mitigation for Transient Execution Attacks
  - Deploying in-memory encryption
  - Data is always encrypted in memory and only decrypted in registers
  - The application self-modifies its instruction and build a trampoline for each one using inline binary rewriting
  - Implementation:
    1. Save scratch registers
    2. Save the effective address
    3. Check secrecy of the address
    4. Compute the shadow address (secret+offset) and emulate the original instruction
    5. Skip addresses falling outside secret region
    6. Restore scratch registers
    7. Emit stolen instructions and jump back



Outline

- Introduction
- Limitations
- Methodology
- Evaluation
- Conclusion

Experimental Results – Detection Effectiveness and Overhead

Table 1: Detection Accuracy

(a) Testbed

	Attack	Fast TPR	Fast FPR	Slow TPR	Slow FPR
AES	F+R	99.99	0.13	99.77	0
	F+F	99.99	0.03	100	0
	P+P	99.99	0.01	100	0
RSA	F+R	99.99	0.13	99.77	0
	P+P	99.99	0.03	100	0
Spectre	PHT	99.99	0.01	100	0
	BTB	99.99	0.13	99.77	0
	RSB	99.99	0.03	100	0
	STL	99.99	0.01	100	0

(b) AWS Fargate

		Fast TPR	Fast FPR	Slow TPR	Slow FPR (%)
AES	F+R	99.99	0.13	99.77	0
	F+F	99.99	0.03	100	0
	P+P	99.99	0.01	100	0
RSA	F+R	99.99	0.13	99.77	0
	P+P	99.99	0.03	100	0.09
Spectre	PHT	99.99	0.01	100	0
	BTB	98.63	0.34	99.92	0
	RSB	99.13	0.26	100	0
	STL	99.64	0.54	99.71	0

PDM shows superior effectiveness for detecting different variants of microarchitectural attacks in both local testbed and cloud.

Table 2: PDM's Overhead on Application Response Time

(a) Real-world Applications

Application	Base (ms)	With PDM (ms) w/o	With PDM (ms) w/
OpenSSH	78.5	80.7	79.1
Tomcat	0.388	0.394	0.389
SQLite	72.62	75.02	72.89

(b) SPEC CPU 2017 Benchmarks

Benchmark	Base (s)	With PDM (s) w/o	With PDM (s) w/
641.leela	883	955	858
648.exchange2	605	653	606
657.xz	3336	3539	3335

PDM introduces negligible overhead in terms of response time and CPU/memory consumption in real world applications and SPEC CPU 2017 benchmark.

Table 3: PDM's CPU/Memory Consumption

Lib	App.					Mem. (MiB)
OpenSSL	AES					43.21
	AES					42.89
	RSA	64	3470	7.3	0.3	35.52
	RSA	256	3718	9.1	0.4	38.43
	ECDH	32	7663	7.7	0.3	42.34
	RSA + AES	288	15215	11.8	3.1	36.67
Libgcrypt	+ ECDH					42.89
	ElGamal	128	4820	7.8	0.3	40.38
	AES	16	4848	9.6	0.3	39.33
	RSA	256	5492	7.6	0.4	40.37
	EdDSA	32	10244	11.6	2.9	41.22

The inference switch can effectively reduce the overhead by preventing unnecessary ML inference.

Experimental Results – Comparison with Existing Works

Table 4: Comparison of PDM and CacheShield [13]

(a) w.r.t. Victim Workload

Workload	PDM		CacheShield	
	TPR (%)	FPR (%)	TPR (%)	FPR (%)
Idle	100	0.12	15.64	0.16
Normal	99.82	0.32	99.13	0.28
High	98.64	0.9	99.86	0.04

3	99.88	0.07	99.87	0.48
5	99.71	0.23	98.24	0.79

Unlike existing HPC-based solutions, PDM can effectively detect attacks under different levels of victim workload and background noise.

Table 5: Comparison of PDM and Cho et al. [20]

P+P	99.78	0.27	99.87	0
Spec.PHT	99.93	0.14	99.8	0

(b) w.r.t. Co-resident Noise

Benign App.	PDM		Cho et al.	
	TPR (%)	FPR (%)	TPR (%)	FPR (%)
O.SSL Speed	99.91	0.12	100	0
Redis	99.94	0.01	100	0
602.gcc	99.84	0.71	100	68.2
600.perlb.	99.87	0.24	100	48.7

Experimental Results – Mitigation Effectiveness and Overhead

Table 6: Effectiveness of Mitigation on Detected Attacks

(a) Cache … on Attacks

Attack	…	…	…	… age (B/s)	Leaked Bytes
F+R/O.SSL …				…46	0
P+P/O.SSL …				…53	0
F+R/Libgc R…				…46	0
P+P/Libgc RSA	0	35	46	Spectre-RSB [28]	116.70	0

PDM's detection can effectively trigger its in-memory encryption mitigation just in time before any leakage occurs.

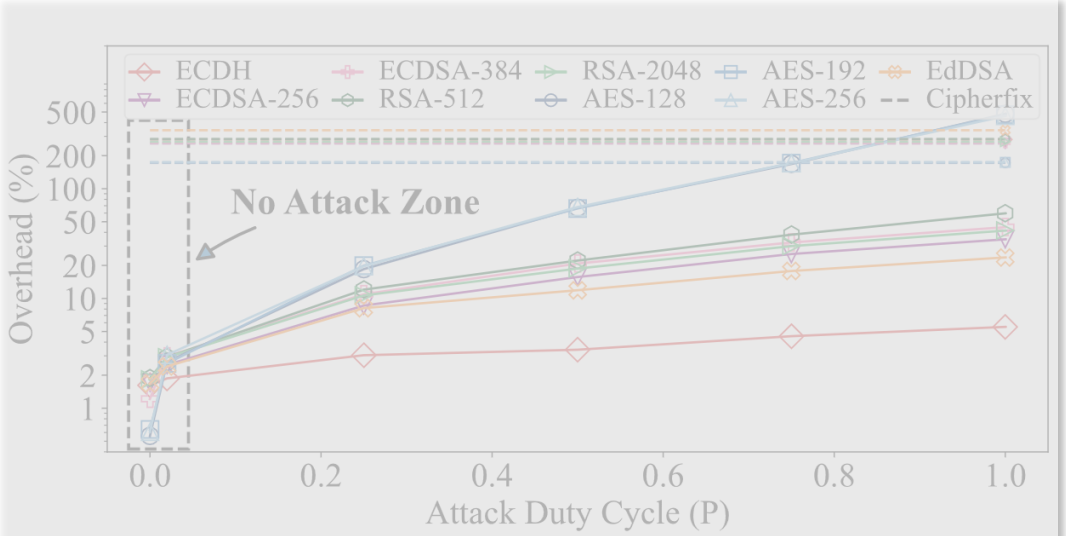
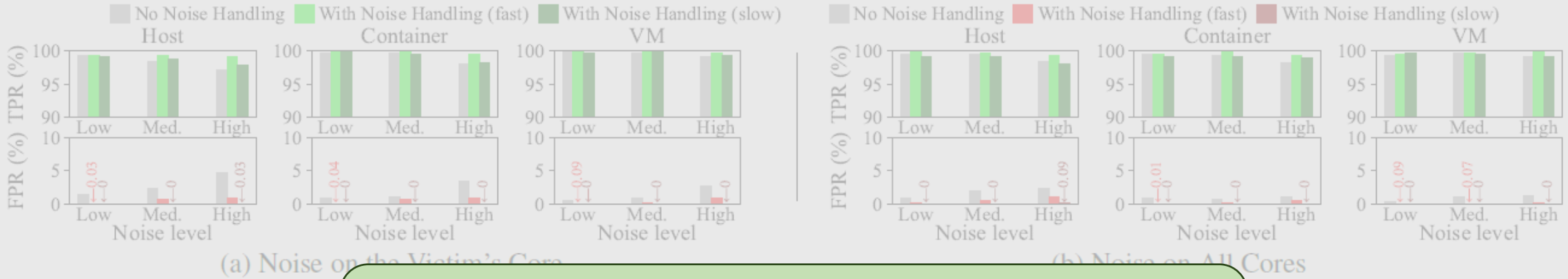


Table 9: PDM's In-memory Encryption Overhead under Different Attack Duty Cycles (P), Workload (λ), and Ratio of Instrumented/Total Memory Accesses, in Comparison to Cipherfix [83] overhead (crypto applications are chosen to allow comparison)

	App.	Base	P=0				…												# of Instrum. Mem. Access	Total # of Mem. Access	Instrum. /Total Ratio (%)	Cipher Fix Over. (%)
			λ = 0.4		λ = 10																	
		Time (μs)	Time (μs)	Over. (%)	Time (μs)	Over. (%)																
OpenSSL	ECDH	556.93	565.94	1.61	566.04	1.63													940K	2.9B	0.03	281
	ECDSA-384	3843.9	3891.2	1.22	3909.1	1.69													62.7M	36.6B	0.17	262
	RSA-512	153.86	156.76	1.88	156.9	1.97													2.9M	2.32B	0.12	284
	RSA-2048	1848.9	1882.7	1.83	1884.4	1.92													56.1M	79.8B	0.07	268
WolfSSL	AES-128	1.51	1.52	0.66	1.52	0.66													347K	2.0M	17.18	172
	AES-192	1.56	1.57	0.64	1.57	0.64	1.6	2.56	1.87	19.87	2.59	66.03	4.23	171.1	8.96	474.3	8.95	473.7	363K	2.1M	17.12	174
	AES-256	1.63	1.64	0.61	1.64	0.61	1.68	3.06	1.95	19.63	2.74	68.09	4.41	170.5	9.13	460.1	9.12	459.5	379K	2.2M	17.08	174
	EdDSA	83.27	84.67	1.68	84.78	1.81	85.27	2.4	90.06	8.15	93.14	11.86	98.06	17.76	102.93	23.6	101.89	22.36	20.7K	102M	0.02	341

PDM's in-memory encryption incurs significantly less overhead in comparison with Cipherfix.

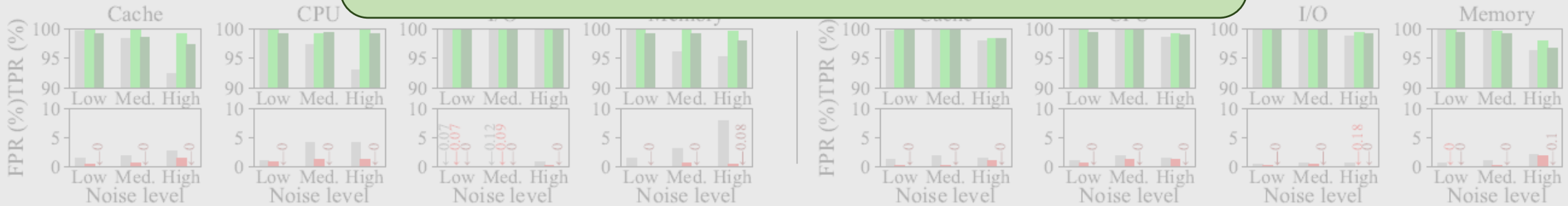
Experimental Results – Impact of Co-resident Noise



(a) Noise on the Victim's Core (b) Noise on All Cores

Figure 8: Impact of Co-resid Isolation), Container-level (cgroups Isolation), VM-le

Enabling noise handling improves the results. Additionally, the slow classifier maintains an almost zero FPR.



(a) Stress on the Victim's Core (b) Stress on All Cores

Figure 9: Impact of Host-level Stressors (Cache, CPU, I/O, and Memory) Noise, with and without Noise Handling

Experimental Results – Impact of Victim Noise and Evasive Attacks

Table 8: Impact of Victim Application Noise

Ap																	
																	FP (%)
O.SSL																	0.2
Libg.																	0
Ng																	0
Redis	99.9	0.4	99.8	0.2	99.6	0	99.7	0.1	99.5	0.1	99.8	0	95.7	2.5	98.6	1.4	99.3	.08
OpenSSH	100	0.6	100	0.6	99.8	0	99.8	0.6	99.8	0.2	99.8	0	98.7	1.8	99.1	0.4	98.3	0

Enabling workload analysis can improve the results. The slow classifier can further reduce false positives in all cases.

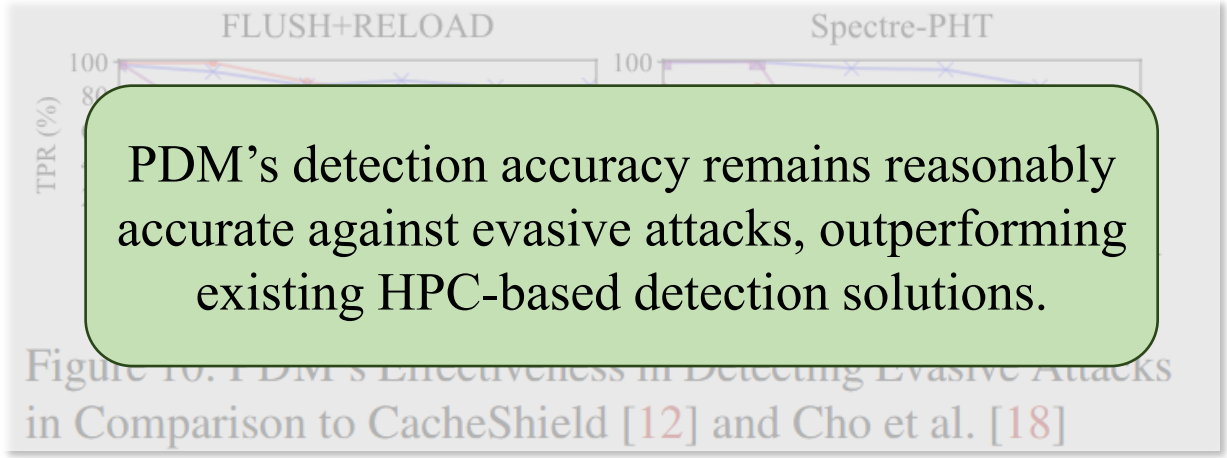


Figure 10: PDM's Effectiveness in Detecting Evasive Attacks in Comparison to CacheShield [12] and Cho et al. [18]

PDM's detection accuracy remains reasonably accurate against evasive attacks, outperforming existing HPC-based detection solutions.

Outline

- Introduction
- Limitations
- Methodology
- Evaluation
- Conclusion

Conclusion

- We proposed PDM to enable cloud tenants to independently detect and mitigate microarchitectural attacks in an accurate, efficient, and timely manner.
- We provided methodologies for (i) probing, (ii) detection, and (iii) mitigation.
- The main benefits of PDM include:
  - Increasing cloud tenants' awareness of microarchitectural attacks
  - Pressuring cloud providers to timely deploy vendor patches
  - Complementing provider-level solutions to provide additional protection

Thank you!

ARC Research Group
https://arc.encs.concordia.ca

Artifact:
https://github.com/arashd021/pdm/