Transit-Guard: An OS-based Defense Mechanism Against Transient Execution Attacks

Maria Mushtaq, David Novo, Florent Bruguier, Pascal Benoit, Muhammad Khurram Bhatti

Transit-Guard: An OS-based Defense Mechanism Against Transient Execution Attacks

Maria Mushtaq, David Novo, Florent Bruguier, Pascal Benoit
LIRMM-CNRS, Univ. Montpellier
Montpellier, France
{maria.mushtaq, david.novo, florent.bruguier, pascal.benoit}@lirmm.fr

Muhammad Khurram Bhatti
Information Technology University (ITU)
Lahore, Pakistan
khurram.bhatti@itu.edu.pk

***Abstract*—Transient attacks manipulate speculative execution to alter the control flow path in an application program and modify microarchitectural state. These state changes are not captured by the existing Instruction Set Architectures (ISAs). In this paper, we propose a novel OS-level detection-based mitigation mechanism, called *Transit-Guard*, that uses machine learning and real-time behavioral data of concurrent processes to detect and subsequently mitigate these attacks at run-time.**

***Index Terms*—Secure Systems, Microarchitecture, Transient execution, Spectre, Meltdown, Mitigation, OS, Machine learning.**

I. The Transit-Guard Mechanism

Side- and covert-channel information leakage is a serious threat to modern computer architectures as they exploit microarchitectural features to extract privileged information. Recently, information retrieval attacks have started to target microarchitectural features beyond the memory sub-system. For instance, many of the performance-boosting techniques included in modern processor microarchitectures, such as out-of-order and speculative execution, pipelining, and branch-prediction [1], have already been exploited. In our threat model, we assume a co-resident attacker process running in user space and targeting privileged address space without explicit access. We consider the OS does not offer any specific privilege level associated with the attacker process.

Transit-Guard works in two distinct stages. As illustrated in Figure 1, the detection module operates in the Linux user space, while the mitigation module operates in kernel space. Transit-Guard works at runtime, i.e., when the attack is actually happening. The Transit-Guard reuses the detection module proposed in [2]. Both Spectre [3] and Meltdown [4] attacks exploit transient execution to trigger the attack and later retrieve data from caches using covert channels. We have selected appropriate hardware/software performance counters (HPCs/SPCs) that are most affected by these stages of the attack. We have selected Total Branch Instructions, Total Branch Mispredictions, L3 total cache misses and total execution cycles as features for Spectre attack. For Meltdown attack, we have selected Total Page Faults, L3 Total Cache Accesses, L3 Cache Misses and Total Execution Cycles as features. As illustrated in Figure 1, the mitigation module is hosted in the Linux kernel space. Once the detection module reports an attack, mitigation module first evaluates whether the received PIDs from detection module are trusted processes or not. All system processes are considered trusted whereas all the user processes are considered untrusted by default. Transit-Guard does so because, at run-time, it is highly likely that the set of active processes also contain some

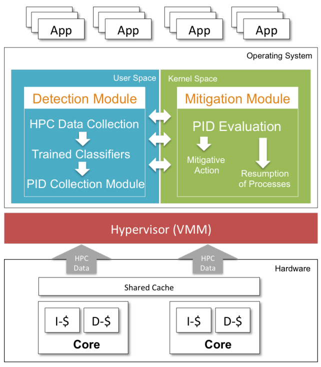

Fig. 1: Design details of the Transit-Guard Mechanism.

Linux's system processes, which are considered as trusted by default. It is, therefore, imperative for the mitigation module to evaluate malicious PIDs. Once it is established that the process under consideration is trusted, the module marks the process as benign/safe. All the untrusted processes are then evaluated further. Since the mechanism associates an instance of each performance counter with processes at the time of their creation, when a process exhibits malicious behaviour the related instance of counter reports abnormality to the detection module, which then acquires the PID of current process. These instances of counters are process-specific and point to the exact process that is exhibiting abnormal behavior at the time of detection. Thus, the mitigation module kills only the malicious process and does not incur any performance overhead other than its own execution time overhead. When a particular model reports a detection, the mitigation module checks the authenticity of the reported detection by waiting for at least 3 consecutive detection reports to further reduce the FPs.

II. Experiments, Results and Discussion

We have performed experiments on Linux Ubuntu LTS 16.04 Kernel version: 4.10.0-28-generic running on Intel's core $i7-4770$ CPU at 3.40-GHz with 64KB L1, 256KB L2, 8192KB L3 and 8GB system memory. We have used Performance API (PAPI) [5] and Perf [6] libraries to access SPCs/HPCs.

A. Case Study 1: Detection and Mitigation of Spectre

1) Detection of Spectre: As illustrated in Table I, our results show that LDA, LR and SVM demonstrate 99.01%, 98.61% and 97.10% detection accuracy, respectively, under FL conditions. All ML models provide high detection accuracy for Spectre attack. We have collected the SPCs/HPCs at a (constant) high speed of $10\mu s$. This sampling frequency is adjustable. Results in Table I illustrate that LDA provides 1.20% and 0.79%, LR provides 1.37% and 0.02% and SVM provides 2.87% and 0.03% of FPs and FNs, respectively. Results demonstrate that all models provide a negligible number of FPs and FNs, where most of the time results depict more FPs in ratio. The adaptability and scalability of detection module is highly dependent on performance overhead. Results in Table I illustrate that all ML models report a low performance overhead for detection, *i.e.*, LDA, LR and SVM report only 1.6%, 1.5%, 1.7% overhead, respectively.

TABLE I: Results on the detection of Spectre attack

Model	Loads	Accuracy (%)	Speed (μs)	FP (%)	FN (%)	Overhead (%)
LDA	NL	99.95	10	0.05	0	1.6
	AL	99.08	10	0.57	0.35	
	FL	98.01	10	1.20	0.79	
LR	NL	99.99	10	0.01	0	1.5
	AL	98.51	10	1.17	0.32	
	FL	98.61	10	1.37	0.02	
SVM	NL	99.30	10	0.69	0.01	1.7
	AL	98.00	10	1.98	0.02	
	FL	97.10	10	2.87	0.03	

TABLE II: Measured time at different stages for mitigation mechanism while detecting Spectre Attack

Load Type	Detection (μs)	PID Collection (μs)	Mitigation (μs)	Total Time (μs)
NL	62	0.5	5	67.5
AL	65	0.5	11	76.5
FL	68	1.0	18	87

2) Mitigation of Spectre: Table II provides results on the measured time at different stages from detection to mitigation, *i.e.*, detection time, PID collection, mitigation and the total time that Transit-Guard takes. The average time Spectre attack takes to execute as stand-alone process on an Intel's Core i7 machine is measured as 256μs. Results in Table II show that Transit-Guard takes 68μs as detection to mitigation time in order to detect an attack process under FL conditions. Once the attack is detected, PIDs of all detected processes are collected within 1.0μs. Once the PIDs of detected processes are collected, the information is relayed to the kernel module through Netlink socket to take mitigation decision, which takes another 18μs to kill the *untrusted* processes. Therefore, Transit-Guard takes 87μs in total under FL conditions to detect and subsequently mitigate Spectre attack. These results demonstrate that Transit-Guard is able to detect and mitigate Spectre attack in $< 27\%$ of attack completion.

B. Case Study 2: Detection and Mitigation of Meltdown

1) Detection of Meltdown: under FL condition, Table III shows that LDA, LR and SVM show 98.30%, 96%, 98.35% detection accuracy, respectively, under FL conditions at the sampling granularity of 10μs. Moreover, LDA offers 1.25% and 0.45%, LR offers 3.40% and 0.60% and SVM offers 1.39% and 0.26% of FPs and FNs, respectively. LDA, LR and SVM incur 1.91%, 2.21% and 2.00% overhead, respectively.

2) Mitigation of Meltdown: Table IV shows the detection to mitigation time taken by the Transit-Guard while mitigating Meltdown attack. It takes 70μs to detect the Meltdown attack, 1.0μs to collect PIDs of detected processes and 21μs to mitigate the *untrusted* processes. Transit-Guard, under FL condition, takes a total of 92μs to detect and mitigate the Meltdown attack. Our experiments show that Meltdown takes 305μs on average to execute the attack on Intel's Core i7 machine. Our results report that Transit-Guard is able to detect and subsequently mitigate Meltdown attack in $< 30\%$ of attack's completion time. Table I & III report that most of the performance overhead is coming from detection module (due to the collection of events and binary classification), whereas, the time to collect PIDs and mitigating *untrusted* process is minimal. Therefore, the overall overhead of Transit-Guard lies between $1.5 - 2.2\%$.

TABLE III: Results on the detection of Meltdown attack

Model	Loads	Accuracy (%)	Speed μs	FP (%)	FN (%)	Overhead (%)
LDA	NL	99.99	10	0.01	0	1.91
	AL	99.91	10	0.09	0.00	
	FL	98.30	10	1.25	0.45	
LR	NL	99.41	10	0.59	0	2.21
	AL	97.45	10	1.95	0.60	
	FL	96.00	10	3.40	0.60	
SVM	NL	99.99	10	0.01	0	2.00
	AL	99.40	10	0.60	0.00	
	FL	98.35	10	1.39	0.26	

TABLE IV: Measured timing at different stages for mitigation mechanism while detecting Meltdown Attack

Load Type	Detection (μs)	PID Collection (μs)	Mitigation (μs)	Total Time (μs)
NL	64	0.5	7	71.5
AL	69	0.5	14	83.5
FL	70	1.0	21	92

III. Conclusion

We propose Transit-Guard, a novel OS-level run-time detection & mitigation mechanism, against transient execution attacks. The Transit-Guard uses multiple machine learning models and profiles concurrent processes using SPCs/HPCs at real-time. Experimental results demonstrate that Transit-Guard is capable of detecting and mitigating Spectre and Meltdown attacks while running under Linux OS. The Transit-Guard is light-weight and resilient to noisy system load conditions.

References

[1] W. Xiong and J. Szefer, "Survey of transient execution attacks," *arXiv preprint arXiv:2005.13435*, 2020.

[2] M. Mushtaq, J. Bricq, M. K. Bhatti, A. Akram, V. Lapotre, G. Gogniat, and P. Benoit, "WHISPER: A tool for run-time detection of side-channel attacks," *IEEE Access*, vol. 8, pp. 83 871–83 900, 2020.

[3] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," *CoRR*, 2018.

[4] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[5] "Performance application programming interface," in $http://icl.cs.utk.edu/papi/$, 2018.

[6] P. Tool, "$http://lacasa.uah.edu/$," 2018.