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

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

To cite this version:

HAL Id: emse-03195702
https://hal-emse.ccsd.cnrs.fr/emse-03195702
Submitted on 14 Apr 2021
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.

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 10µs. 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

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

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

<table>
<thead>
<tr>
<th>Load Type</th>
<th>Detection (µs)</th>
<th>PID Collection (µs)</th>
<th>Mitigation (µs)</th>
<th>Total Time (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NL</td>
<td>62</td>
<td>0.5</td>
<td>5</td>
<td>67.5</td>
</tr>
<tr>
<td>AL</td>
<td>65</td>
<td>0.5</td>
<td>11</td>
<td>76.5</td>
</tr>
<tr>
<td>FL</td>
<td>68</td>
<td>1.0</td>
<td>18</td>
<td>87</td>
</tr>
</tbody>
</table>

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.

### TABLE III: Results on the detection of Meltdown attack

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

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

<table>
<thead>
<tr>
<th>Load Type</th>
<th>Detection (µs)</th>
<th>PID Collection (µs)</th>
<th>Mitigation (µs)</th>
<th>Total Time (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NL</td>
<td>64</td>
<td>0.5</td>
<td>7</td>
<td>71.5</td>
</tr>
<tr>
<td>AL</td>
<td>69</td>
<td>0.5</td>
<td>14</td>
<td>83.5</td>
</tr>
<tr>
<td>FL</td>
<td>70</td>
<td>1.0</td>
<td>21</td>
<td>92</td>
</tr>
</tbody>
</table>

### 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**


