DAWG: A Defense Against Cache Timing Attacks in Speculative Execution Processors

Submitted by – CS19D008 & CS19S016

1. Insight to the paper

DAWG (Dynamically Allocated Way Guard) is a mechanism that securely partition cache ways into different protection domains. This partitiong provides strong isolation between protection domains by fully isolating hits, misses and metadata accross different protection domains. DAWG was proposed as a defense against Cache timing attacks in Speculative Execution Processors.

1.1 Attack domains

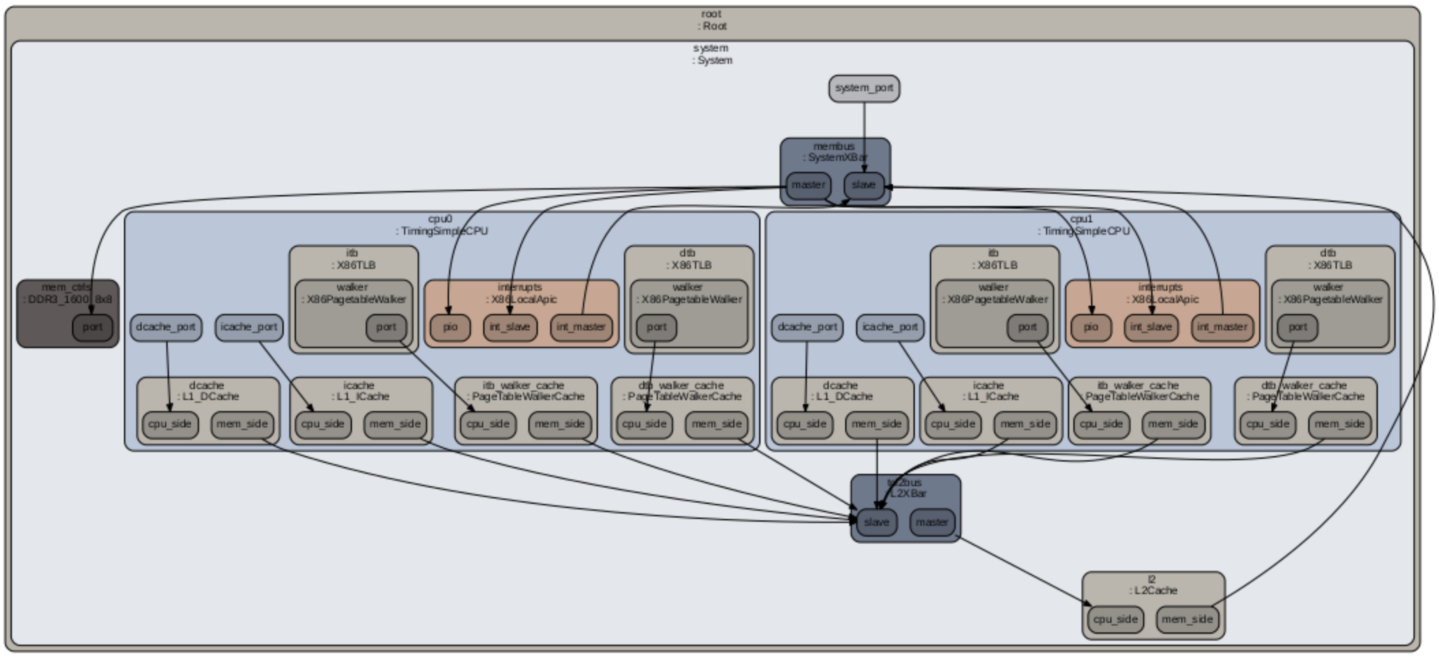
Attacks like flush+reload, prime+probe, evict +reload etc use channels at various levels of the memory cache hierarchy and exploit cache lines shared between an attacker's program and the victim process to exfilterate information. The basic idea used in all these attacks is that, two entities can communicate on a trusted environment by modulating the use of shared cache. The attacker can gather information indirectly by observing conflict misses caused by the victim’s access.

1.2 High-level design

A conventional set associative cache is divided into different sets and every set consist of a number of ways. A cache may be allocated in any way in a particular set. DAWG is implemented by allocating a group of ways to protection domains which restricts the allocation, hits, misses, replacements etc to the ways allocated to the protection domain. This restriction isolates the visibility of any cache accesses to a single protection domain which in turn preclude the existence of any cache state based channels between the protection domains of attacker and receiver. This restricts any leakage of information from the victim to the attacker.

2. Implementation

2.1 System



2 core system simulated

We have simulated a 2 core system in Gem5 for implementing DAWG, which is shown in the figure above. The cache configuration is as given below.

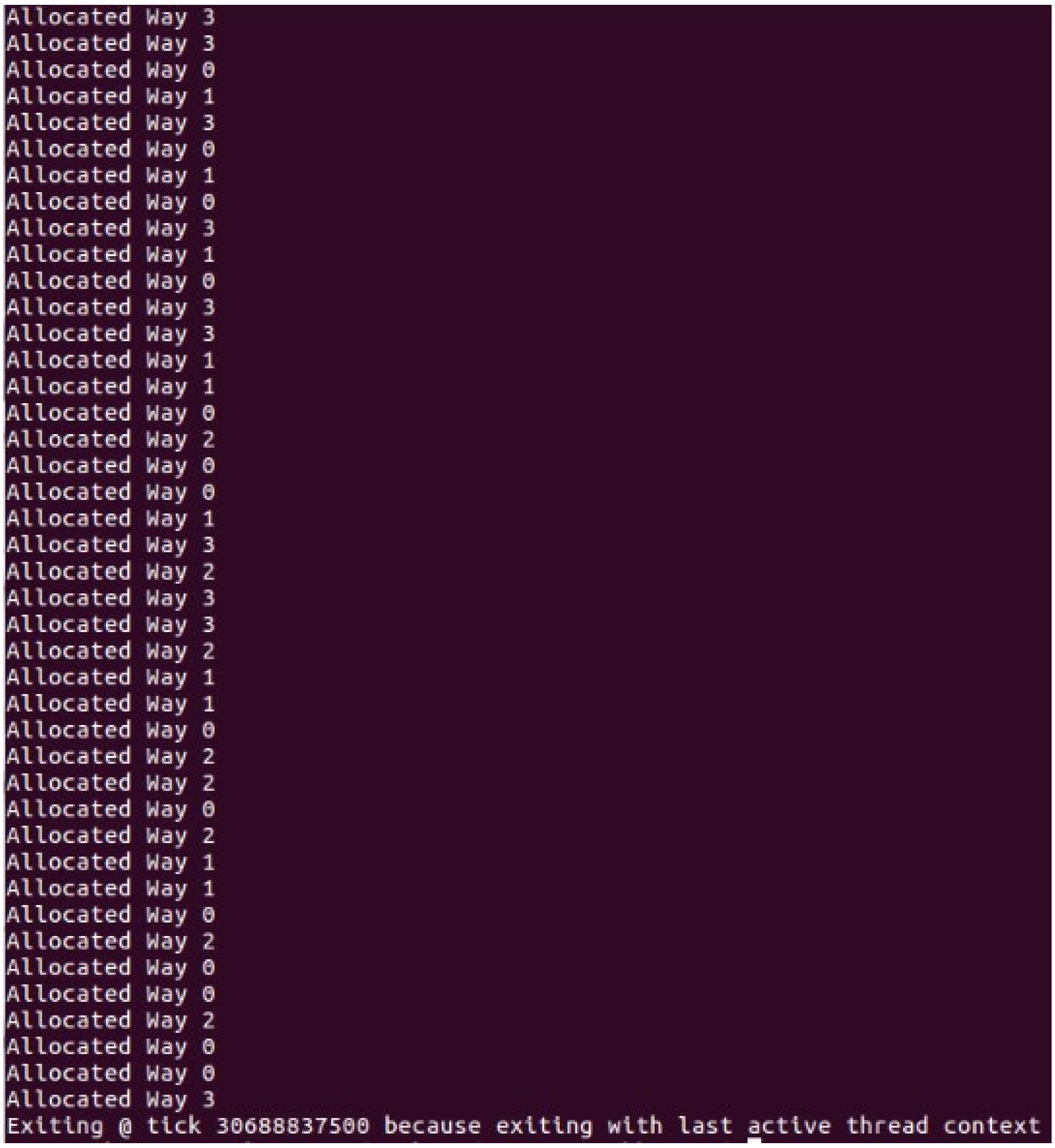
Private L1 Data Cache with size = 32kB and associativity = 4

Private L1 Instruction Cache size= 32kB and associativity = 4

Shared L2 Cache size = 64kB and associativity = 4

2.2 Conventional Cache

We have executed a sample program and the cache ways allotted to them is shown in the following screenshot. It can be observed that in a conventional cache, all the ways in a set will be used for every execution of a program.



```
Allocated Way 3
Allocated Way 3
Allocated Way 0
Allocated Way 1
Allocated Way 3
Allocated Way 0
Allocated Way 1
Allocated Way 0
Allocated Way 3
Allocated Way 1
Allocated Way 0
Allocated Way 3
Allocated Way 3
Allocated Way 1
Allocated Way 1
Allocated Way 0
Allocated Way 2
Allocated Way 0
Allocated Way 0
Allocated Way 1
Allocated Way 3
Allocated Way 2
Allocated Way 3
Allocated Way 3
Allocated Way 2
Allocated Way 1
Allocated Way 1
Allocated Way 0
Allocated Way 2
Allocated Way 2
Allocated Way 0
Allocated Way 2
Allocated Way 1
Allocated Way 1
Allocated Way 0
Allocated Way 2
Allocated Way 0
Allocated Way 0
Allocated Way 2
Allocated Way 0
Allocated Way 0
Allocated Way 3
Exiting @ tick 30688837500 because exiting with last active thread context
```

Ways allotted in a conventional 4 way associative cache

2.3 Way partitioned cache

The default cache implementation in Gem5 is modified to incorporate the way partitioning. The modifications are done in the following logics.

2.3.1. Hit/Miss Isolation

In a conventional set associative cache a read access would return a cache line to the requested core if its a match (hit) in any of the way comparisons. This allows the attacker to gather some information by probing the particular set. So hits in the DAWG cache must be isolated so that a cache access must not hit in ways it was not allocated.

If the access is not a hit, the requested block has to be brought to the cache by replacing any of the block in the set. Here the victim selection is also restricted to only the ways allocated to the protection domain. This ensures that the hits and misses of a program running in one protection domain are not affected by program behavior in different protection domains.

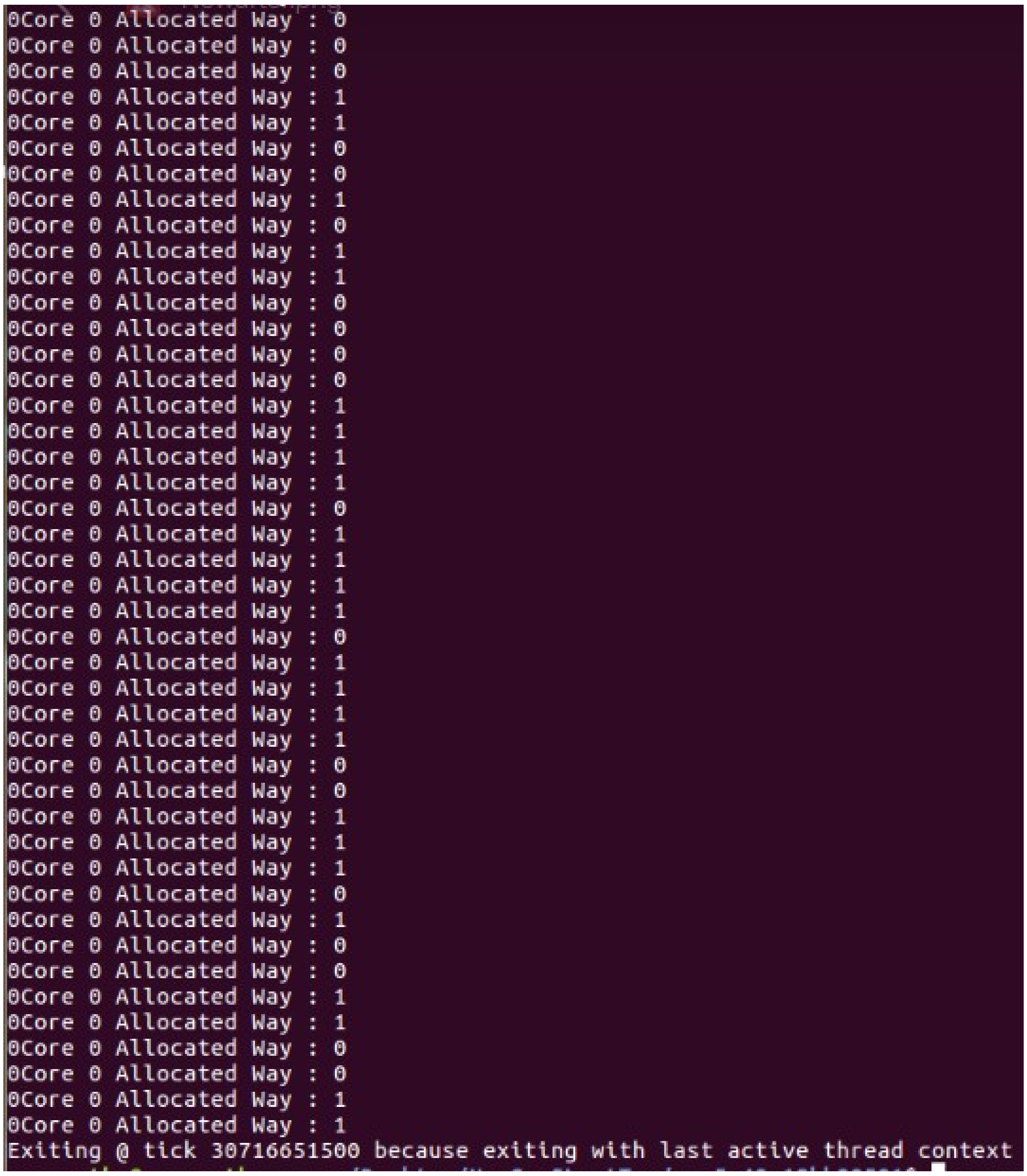
2.3.2 Allocation of ways to different protection domains

A subset of ways are allotted to different protection domains based on which core the program is executed. So programs executed in different cores will be given different subset of ways which makes it difficult for the attacker running in a different core to probe a particular cache set to mount the attack.

2.3.3 Victim Selection and replacement

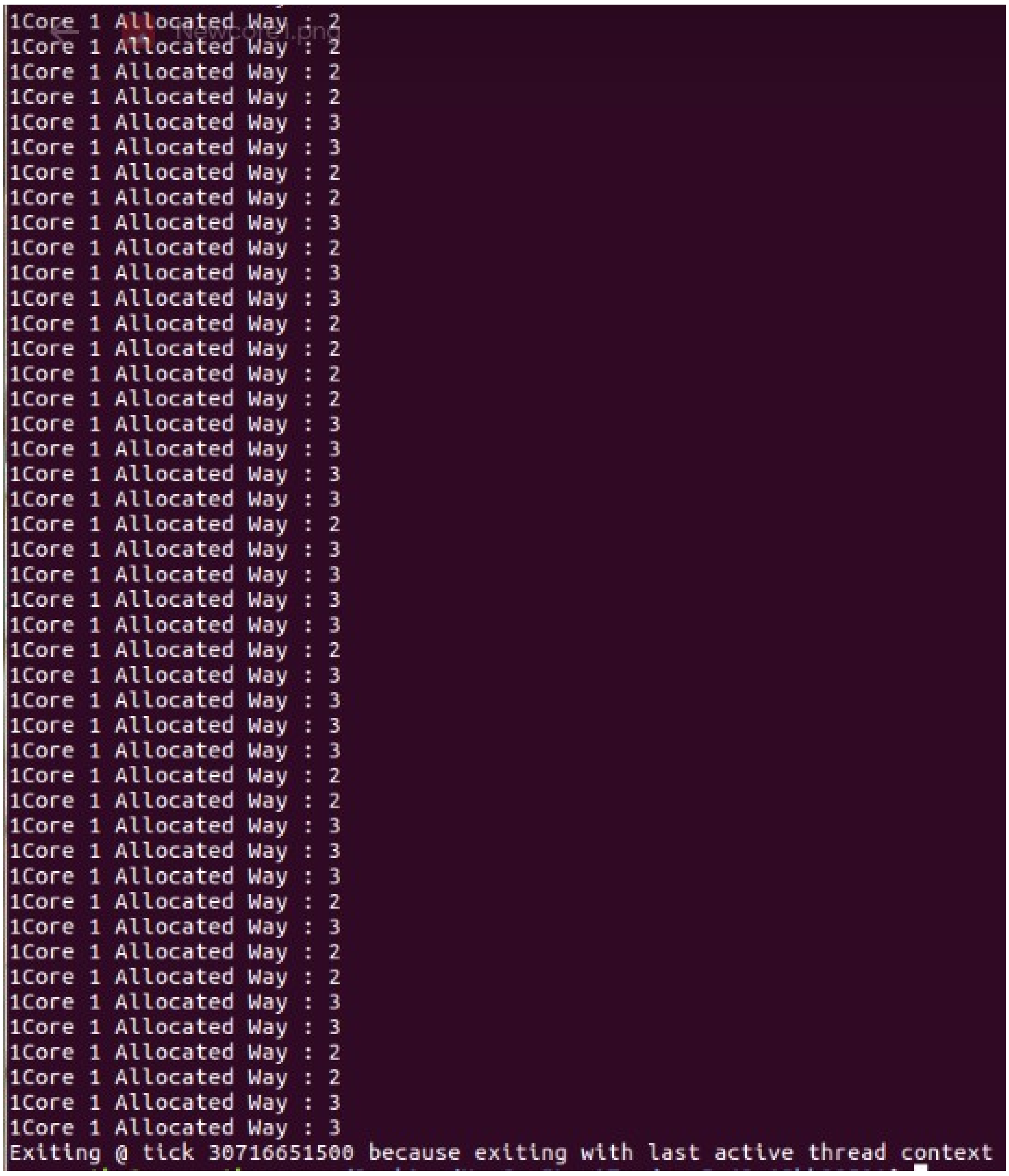
The victim selection chooses a victim among the subset of ways allotted to a particular domain which ensures that eviction from a particular domain is possible only by the same domain. Replacement strategy used here is an LRU based scheme which is same as that of the conventional cache.

3. Results



```
0Core 0 Allocated Way : 0
0Core 0 Allocated Way : 0
0Core 0 Allocated Way : 0
0Core 0 Allocated Way : 1
0Core 0 Allocated Way : 1
0Core 0 Allocated Way : 0
0Core 0 Allocated Way : 0
0Core 0 Allocated Way : 1
0Core 0 Allocated Way : 0
0Core 0 Allocated Way : 1
0Core 0 Allocated Way : 1
0Core 0 Allocated Way : 0
0Core 0 Allocated Way : 0
0Core 0 Allocated Way : 0
0Core 0 Allocated Way : 0
0Core 0 Allocated Way : 1
0Core 0 Allocated Way : 1
0Core 0 Allocated Way : 1
0Core 0 Allocated Way : 1
0Core 0 Allocated Way : 0
0Core 0 Allocated Way : 1
0Core 0 Allocated Way : 1
0Core 0 Allocated Way : 1
0Core 0 Allocated Way : 1
0Core 0 Allocated Way : 0
0Core 0 Allocated Way : 1
0Core 0 Allocated Way : 1
0Core 0 Allocated Way : 1
0Core 0 Allocated Way : 1
0Core 0 Allocated Way : 0
0Core 0 Allocated Way : 0
0Core 0 Allocated Way : 1
0Core 0 Allocated Way : 1
0Core 0 Allocated Way : 1
0Core 0 Allocated Way : 0
0Core 0 Allocated Way : 1
0Core 0 Allocated Way : 0
0Core 0 Allocated Way : 0
0Core 0 Allocated Way : 1
0Core 0 Allocated Way : 1
0Core 0 Allocated Way : 0
0Core 0 Allocated Way : 0
0Core 0 Allocated Way : 1
0Core 0 Allocated Way : 1
Exiting @ tick 30716651500 because exiting with last active thread context
```

Programs running on core 0 is allotted with ways 0 and 1



```
1Core 1 Allocated Way : 2
1Core 1 Allocated Way : 2
1Core 1 Allocated Way : 2
1Core 1 Allocated Way : 2
1Core 1 Allocated Way : 3
1Core 1 Allocated Way : 3
1Core 1 Allocated Way : 2
1Core 1 Allocated Way : 2
1Core 1 Allocated Way : 3
1Core 1 Allocated Way : 2
1Core 1 Allocated Way : 3
1Core 1 Allocated Way : 3
1Core 1 Allocated Way : 2
1Core 1 Allocated Way : 2
1Core 1 Allocated Way : 2
1Core 1 Allocated Way : 2
1Core 1 Allocated Way : 3
1Core 1 Allocated Way : 3
1Core 1 Allocated Way : 3
1Core 1 Allocated Way : 3
1Core 1 Allocated Way : 2
1Core 1 Allocated Way : 3
1Core 1 Allocated Way : 3
1Core 1 Allocated Way : 3
1Core 1 Allocated Way : 3
1Core 1 Allocated Way : 2
1Core 1 Allocated Way : 3
1Core 1 Allocated Way : 3
1Core 1 Allocated Way : 3
1Core 1 Allocated Way : 3
1Core 1 Allocated Way : 2
1Core 1 Allocated Way : 2
1Core 1 Allocated Way : 3
1Core 1 Allocated Way : 3
1Core 1 Allocated Way : 3
1Core 1 Allocated Way : 2
1Core 1 Allocated Way : 3
1Core 1 Allocated Way : 2
1Core 1 Allocated Way : 2
1Core 1 Allocated Way : 3
1Core 1 Allocated Way : 3
1Core 1 Allocated Way : 2
1Core 1 Allocated Way : 2
1Core 1 Allocated Way : 3
1Core 1 Allocated Way : 3
Exiting @ tick 30716651500 because exiting with last active thread context
```

Programs running on core 1 is allotted with ways 2 and 3

3.1 Performance statistics

System	Number of hits	Number of misses	Latency
Conventional cache	9093400	2837	234174500
Core 0 Partioned cache	9093164	3073	260146500
Core 1 Partioned cache	9093164	3073	260146500

4. Observation and conclusion

It can be observed that, for the conventional cache the number of hits is more and the number of misses are less which leads to less latency and more performance.

But for the partitioned cache, the hits are less and misses are more comparing that of the conventional cache. This leads to increased latency and decreased performance in DAWG than that of the conventional cache. Eventhough the performance is reduced, it ensures more security by restricting the cache accesses to protection domains.

5. References

[1] Vladimir Kiriansky, Ilia Lebedev, Saman Amarasinghe, Srinivas Devadas, and Joel Emer. “DAWG: A defense against cache timing attacks in speculative execution processors” in International Symposium on Microarchitecture. IEEE, 2018.

[2] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The GEM5 Simulator," SIGARCH Computer Architecture News, vol. 39, no. 2, May 2011.

[3] http://gem5.org/Main_Page last accessed on 17/11/2019

[4] http://learning.gem5.org/ last accessed on 17/11/2019