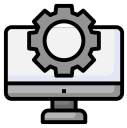
Advanced Computer Architecture



HW1

Prof :

Andreas Moshovos

Student :

Hooman Keshvari

Student Number :

1011293869

Question 1

1. (a) The key challenge is that previous cache side-channel attacks relied on L1 cache which is practically small to probe and also infeaseble due to each core having its own distinguished L1 cache in an environment like the cloud but the last level cache is shared across all cores but it is slower in terms of access time which make previous approaches ineffective.
The goal is to actually develop an effective side-channel attack across virtual machines on the cloud which relies on the shared last level cache between these virtual machines regardless of special conditions like shared memory or VMM level vulnerabilities.

   (b) Acording to the **CEASER** paper:

   > the group of lines that map to the same set of the cache and can cause an eviction is called an Eviction Set

   This basically means that eviction set is the collection of memory segments that when brought to the cache, would replace a certain line and throw out whatever is in that line (flush that line of cache)
   Now Let us analyze **Algorithm 1** :

   - First foreach (foreach *candiate* $\in$ lines ...): This code will probe each candidate to see whether it conflicts with any members of the conflict set and inputs it if there is no conflict. Output of this loop is actually the initial conflict set which are candidates with no conflicts. So this loop wants to actually collect a set of memory lines that dont evict eachother from the cache until there is no more non-conflicting lines.
   - The second loop is basically checking candidates which are in lines other than the conflict set and if the candidate has any conflicts with any of the members of conflict set, then the inner loop is to find out what element in the conflict set was actually conflicting with our candidate and inserting that into the eviction set and taking it out of the conflict set.
     Now in the inner foreach loop we do not process all candidates as we already know there is a conflict between that candidate and one member in conflict set and we know that no too members of conflict set would conflict so this one candidate is enough.

2. - APLR is the rate at which CEASER remaps lines of a certain set in the cache which means that if our APLR is K, then each K*W accesses to the cache would cause a set to get remapped. So if the APLR is set to 200 in a 16-way cache, it means that every 200*16 access to the cache would cause a set to be remapped. In the paper we could see that for an APLR=100 there is an approximate 1% overhead so if we use APLR=200, since we would have half as much remapping, then we could say that the overhead would become less. This remapping overhead does not directly translate to slowdown because cache hits can still occur while remapping occurs and also not all remapped lines cause conflicts (This fact is shown in the paper where APLR=100 has 1% remapping overhead but only 0.74% actual slowdown). Like for example in some workloads like a streaming service, evictions are less likely to impact performance so remapping also would not introduce any significant penalties as opposed to databases which this remapping could have a costly performance impact.
   - So for a 1MB bank of LLC with 16-way associativity, the linesize is 64 bytes which makes the total number of lines 1MB/64B = 16384 which also means the total number of sets is 16384/16 =1024. According to their analysis you need F=0.42 of lines for an attack which means that L = F * N = 0.42 * 16384 = 6881 lines. Now you need $\frac{R*L^2}{2}$ accesses

for this attack which the paper presumes R=2, so the total accesses needed would be $\frac{2*6881^2}{2} = 47.3 * 10^6$ accesses. Now the listed cache latency is 24 cycles and their config is at 3.2GHz which makes 7.5ns for each cache access. This means that the total amount of time needed is $47.3 * 10^6 * 7.5 * 10^{-9} = 0.35$ which is almost the same as their number in the table(0.4)

- – tCAS : Column Address Strobe latency which is the time between sending a column address and receiving the data
  - tRCD : Row to Coumn Delay which is the time needed between activating a row and accessing columns in that row
  - tRP : Row Precharge which is the time needed to precharge a row before another row would be activated
  - tRAS : Row Access Strobe which is the minimum a row must remain open for a read or write to complete
- In the hinted paper there is a phenomenon discussed where different mapping functions can potentially reduce conflicts between cache lines so there could be a case where a remapping could create a better mapping than the original one. Now in the hinted paper, we see that their design helps avoid having neighbor lines conflict for same cache locations which is basically the same thing CEASER does but with a different method. So although CEASER would decrease the perfomance a little bit in average case but there would be cases that it unintentionally would create better mappings for better cache utilization and fewer conflicts for certain workloads.

Question 2

1. Gadget is basically a piece of code which is used in attacks. This piece of code is available in the memory of the victim and is usually found in shared libraries or victim's own codes which the attacker would need to find its sequences. One example is an attack called ROP(return oriented programming). Like look at the piece of code bellow :

```
add $a1, $a1, 1
ret
```

   this is a very simple piece of assembly which could be found in almost every program but we could use this as a tool and having enough of these and having control over the stack means that the attacker could basically execute their code in the victim's memory.

2. a sample of a gadget which spectre mentions is as below :

```
adc edi,dword ptr [ebx+edx+13BE13BDh]
adc dl,byte ptr [edi]
```

   The found this gadget on both windows 8 and windows 10 in ntdll.dll where if the attacker controls ebx and edi registers, it would allow the attacker to read the victim's memory