Assignment 8 (Semil Jain - u1417989)

1. Security
Soln:

Spectre Attack:

The basic principle of spectre attack is based on that of the Meltdown attack. Here, Instead of solely relying on the attacker code, The attacker plays with the victim code bugs and uses that to their advantage.

i) So the Attacker and the Victim both reside in the same server and execute in this same server. This indicates that they share multiple resources like Caches and Branch Predictors.

ii) Now, as previously mentioned, there is a piece of unsafe stanza in the victim code which is similar to what we saw in Meltdown (lw R1 <- [illegal address] , lw … <- [R1]). The victim code can be something like y = array2[array1[x]]; Here array1 is the secret array and the way array 2 accesses the memory gives us the foot print to the secret.

iii) In variant 1, Suppose we have a common web server, The attacker can control the input via its own queries and based on the input control, Pieces of code like mentioned above can bring unwanted secrets into the cache. The attacker can cause an exception/ mispredict condition, but due to the hardware design bug that allows the rest of the instructions to keep on executing Our secret value is used to bring a random set into the cache. Since cache are speculative values and don't get squashed, The attacker set a Prime at the start by bringing their own data into the cache like a snow cover. At a later point, after the above buggy sequence happens and the attacker probes the cache using their set trap, They find the access times of bringing bits into cache and long access times indicate that they were brought from memory. Maybe this can lead to a lot of noisy data as well, But using brute force the attacker can try to figure out the secret in some way. If the victim is smart and wants to prevent this, they can write their access code in a different way based on a condition which would lead to array access much later in time. But as mentioned in spectre variant 2, The attacker knows the victim code and tries to implement the same branching strategy so as to use the same shared branch predictor as the victim and jump to the same label to steal the info in the above mentioned way.

iv) With the help of the above concepts, like out of order execution, branch prediction etc., The attacker can now provide any value of the memory that they need access and the victim code may bring it in even though they don't have access. Instead of squashing the remaining instructions the processor lets it to execute and we can amplify this to access anything in the memory, maybe 8 bits at a time.

2. Snooping-Based Cache Coherence

Assume that X and Y are not in any of the caches at the start of the sequence, the caches are direct-mapped, and blocks X and Y map to the same set in each cache (X and Y cannot co-exist in a cache at any time)

Request	Cache Hit/Miss	Request on the bus	Who responds	State in Cache 1	State in Cache 2	State in Cache 3
				Inv	Inv	Inv
P1: Write X	Miss	Write X	Memory	M	Inv	Inv
P2: Write X	Miss	Write X	P1	Inv	M	Inv
P3: Read X	Miss	Read X	P2(Mem Wrtbck)	Inv	S	S
P1: Read X	Miss	Read X	Memory	S	S	S
P3: Write X	Perm Miss	Upgrade X	No one (Other caches Invalidate)	Inv	Inv	M
P3: Read Y	Miss	Read Y	Memory(Evict X and Mem Wrtbck X)	Inv	Inv	S
P2: Write Y	Miss	Write Y	Memory	Inv	M	Inv

3. Directory-Based Cache Coherence

Assume that a message can include some control information as well as an address and cache line. Also assume that the home nodes for memory locations X and Y are both associated with processor P4. Assume that X and Y are not in any of the caches at the start of the sequence, the caches are direct-mapped, and blocks X and Y map to the same set in each cache (X and Y cannot co-exist in a cache at any time).

Request	Cache Hit/Miss	Messages	Dir State (P4)	State in C1	State in C2	State in C3	State in C4
				Inv	Inv	Inv	Inv
P1: Write X	Miss	Wr-req to Dir. Dir responds.	X: M: 1	M	Inv	Inv	Inv
P2: Write X	Miss	Wr-req to Dir. Dir fwds request to P1. P1 sends data to Dir. Dir sends data to P2	X: M: 2	Inv	M	Inv	Inv
P3: Read X	Miss	Rd-req to Dir. Dir fwds request to P2. P2 sends data to Dir. Memory wrtbk. Dir sends data to P3.	X: S: 2, 3	Inv	S	S	Inv
P1: Read X	Miss	Rd-req to Dir. Dir responds	X: S: 2, 3, 1	S	S	S	Inv
P3: Write X	Perm Miss	Upgr-req to Dir. Dir sends INV to P2 and P1. P2 and P1 sends ACK to Dir. Dir grants perms to P3.	X: M: 3	Inv	Inv	M	Inv
P3: Read Y	Miss	Rd-req to Dir. Dir responds (X is evicted and Writeback happens)	Y: S: 3	Inv	Inv	S	Inv
P2: Write Y	Miss	Wr-req to Dir. Dir sends INV to P3. P3 sends ACK to Dir. Dir grants perms to P2.	Y: M: 2	Inv	M	Inv	Inv

Write invalidate happens when someone is reading a data and the next instructions comes in to modify that data. The directory that instructs all readers to invalidate their data and on receiving ack, gives write permission. While performing a write invalidate (excluding the write request and giving the write permission after invalidation) There are two requests going back and forth per reader. **In our case total such requests are 6.** (Four to P1 and P2 while P3: Write X and two to P3 while P2 Write Y)