Lincoln Sand

1. In a cache side channel attack, how does the victim leave a secret-dependent footprint in the cache? How does the attacker extract this secret? (20 points)

This is a rough description of meltdown: The attack is done by the attacker first warming up the cache (filling it up with their own data). The victim application then speculatively fetches data based on the supplied value by the attacker. The processor will fetch the data before the priviledge check is finished (if the priviledge check fails, it just squashes the result). But even though the result was squashed, it resulted in a cache entry being invalidated. The attacker then iterates through their memory they warmed the cache with and based on which parts result in cache misses, they can deduce what the secret value that was used in the speculative fetch was.

As long as the victim application does a memory read using a secret-dependent value, the attacker can examine the cache misses/access patterns to recover it.

2. Answer the following questions about virtual/-physical memory.

a. Does the program code produce virtual or physical addresses? (5 points)

It produces virtual addresses.

b. Why is the program's virtual memory address converted into a physical memory address? (5 points)

Having virtual memory is great because you can 'fake' having more memory than you have. It also appears to be contiguous, which it might not be in actual physical memory (allowing non-contiguous pages in physical RAM allows for less waste/internal fragmentation). To elaborate on 'faking more memory', you can swap part of the application's memory to disk and the program will not be able to tell (except for the performance of the memory access page fault). It also allows for program isolation, which is nice for security, since it stops one process from mucking around in the memory of another. Also, it allows you to load processes into memory exactly the same without having to worry about which parts are already being used.

c. How does the hardware do an efficient translation from virtual to physical address? (10 points)

The hardware has a MMU (Memory Management Unit) which translates the virtual memory to physical memory automatically based on the entries in the paging structure (which is usually set by the OS). To speedup the translation, recent mappings are cached in the TLB (Translation Lookaside Buffer). So, the MMU first checks the TLB for the mapping and uses it if found. But if not found, it then has to access the paging structure to do the address lookup and then update the TLB to contain this most recent mapping.

3. Consider a 3-processor multiprocessor connected with a shared bus that has the following properties: (i) centralized shared memory accessible with the bus, (ii) snooping-based MSI cache coherence protocol, (iii) write-invalidate policy. Also assume that the caches have a writeback policy. Initially, the caches all have invalid data. The processors issue the following five requests, one after the other. Create a table similar to that in slide 6 of lecture 26 to indicate what happens for every request. (40 points)

1. P3: Read X
2. P1: Read X
3. P1: Write X
4. P3: Write X
5. P2: Read X

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
P3: Read X	Rd Miss	Rd X	Memory	Inv	Inv	S
P1: Read X	Rd Miss	Rd X	Memory	S	Inv	S
P1: Write X	Perms Miss	Upgrade X	No response. Other caches invalidate.	M	Inv	Inv
P3: Write X	Wr Miss	Wr X	P1 responds.	Inv	Inv	M
P2: Read X	Rd Miss	Rd X	P3 responds. Mem wrtbk.	Inv	S	S

4. Read some of the following blog articles about interesting problems in computer architecture. Write a 100-200 word paragraph describing what you learnt from one of these articles. (20 points)

I'm going to write about "Computer Architecture Research In Space":

This article describes the fact that satellites have moved from monolithic, large, expensive satellites, with significant budgets, to small inexpensive networks of satellites built of off-the-shelf components. It describes how both the power constraints introduced from shrinking the solar panel size, as well as the bandwidth and computational constraints, is a largely under addressed and ripe area for computer architecture to address. The author also reframes the problem as one of edge computing: distributing the work across multiple nanosatellites and trying to avoid sending down to earth as much data as possible. In other words, the satellites try to do as much computation as possible onboard to avoid having to send extraneous data to earth over constrained and unreliable data signals. This leads to multiple problems: trying to adapt machine learning to energy and time constrained systems (as there is a fixed amount of time between frames to process data), managing and efficiently distributing work across a distributed system of nanosatellites (including training of machine learning models), and rationing energy needs (especially as it relates to attitude control).