

Lecture 25: Security, VM

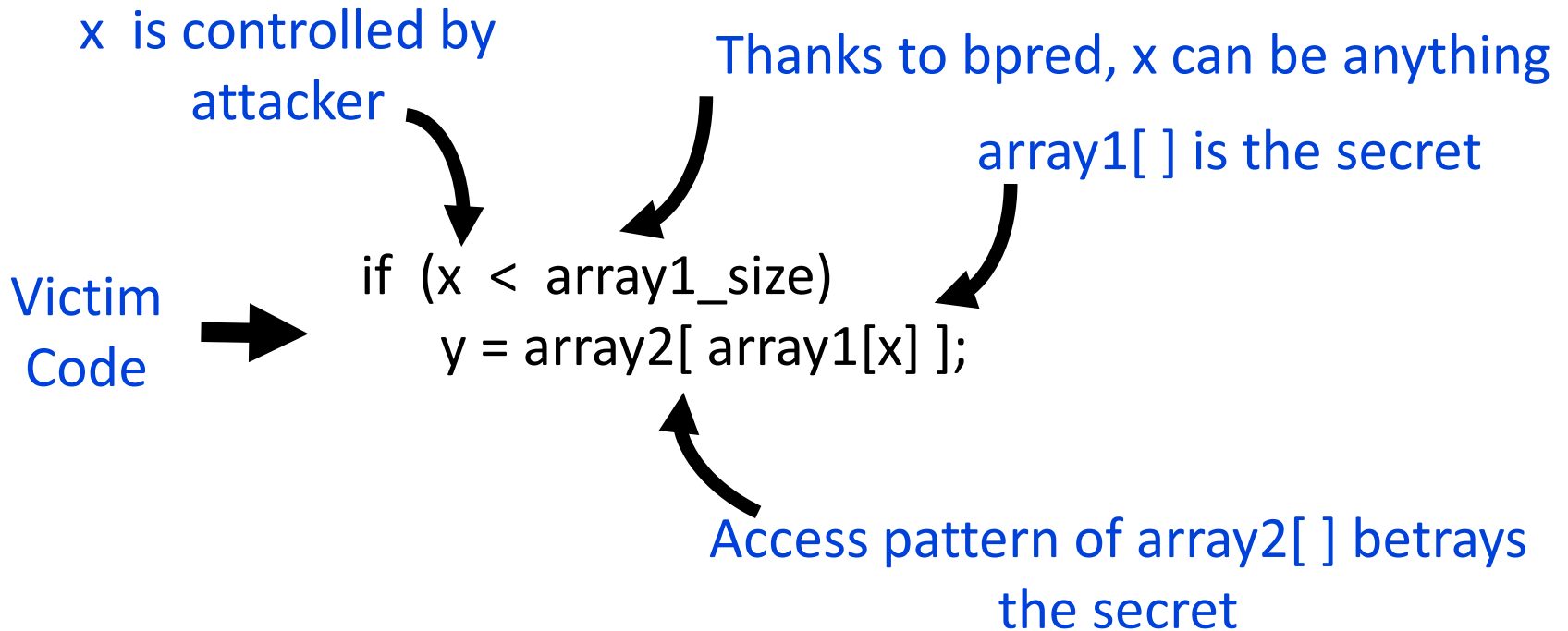
- Today's topics:
 - Security
 - Virtual memory

Hardware Security

- Software security: key management, buffer overflow, etc.
- Hardware security: hardware-enforced permission checks, authentication/encryption, etc.
- Information leakage, side channels, timing channels
- Meltdown, Spectre, SGX

Meltdown

Spectre: Variant 1



Spectre: Variant 2

Attacker code

Label0: if (1)

Label1: ...

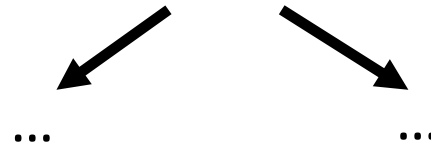


Victim code

R1 ← (from attacker)

R2 ← some secret

Label0: if (...)



Victim code

Label1:

lw [R2]

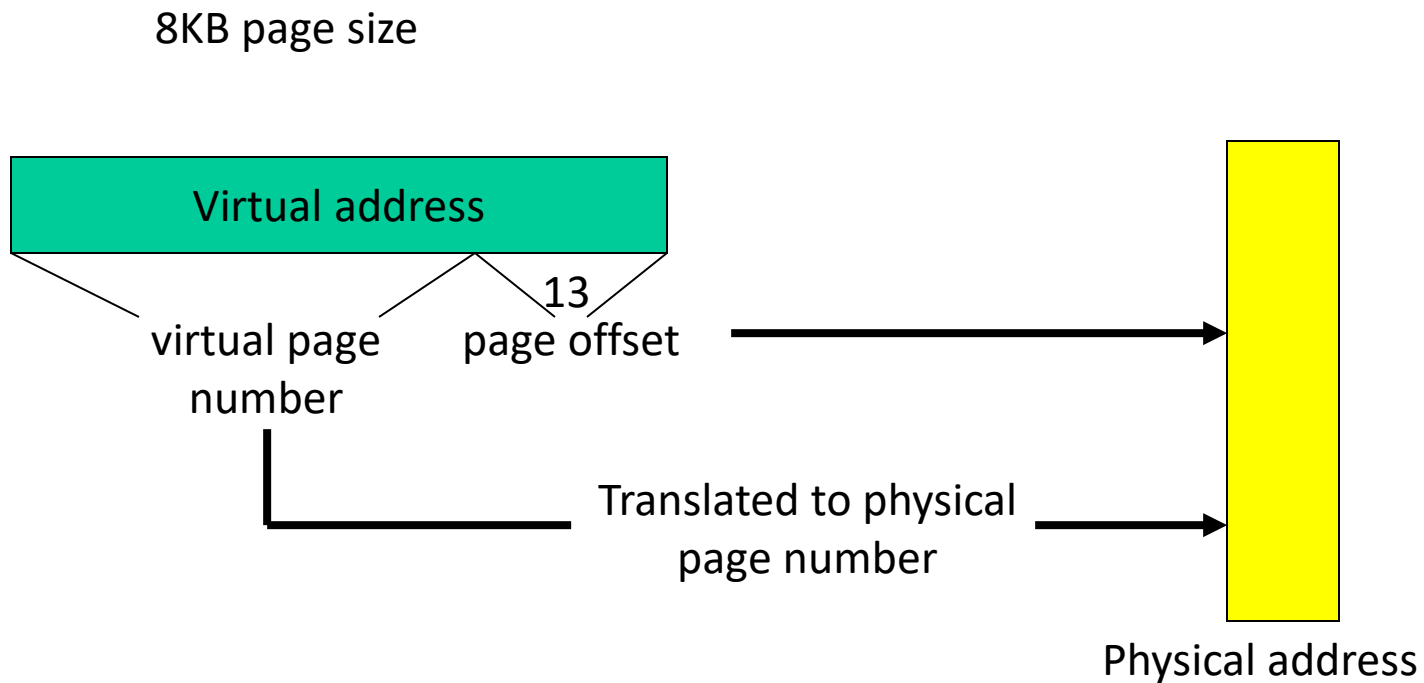
Virtual Memory

- Processes deal with virtual memory – they have the illusion that a very large address space is available to them
- There is only a limited amount of physical memory that is shared by all processes – a process places part of its virtual memory in this physical memory and the rest is stored on disk (called swap space)
- Thanks to locality, disk access is likely to be uncommon
- The hardware ensures that one process cannot access the memory of a different process

Virtual Memory

Address Translation

- The virtual and physical memory are broken up into pages



Memory Hierarchy Properties

- A virtual memory page can be placed anywhere in physical memory (fully-associative)
- Replacement is usually LRU (since the miss penalty is huge, we can invest some effort to minimize misses)
- A page table (indexed by virtual page number) is used for translating virtual to physical page number
- The page table is itself in memory

TLB

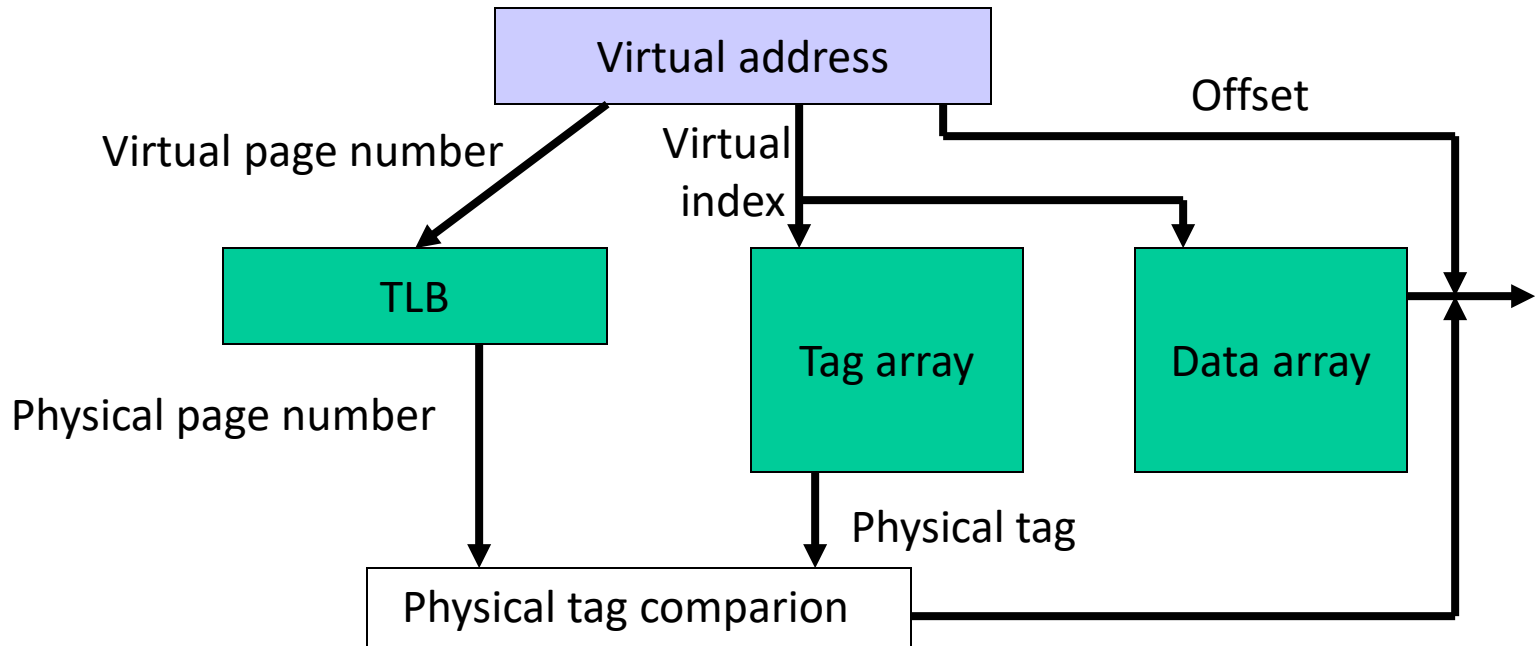
- Since the number of pages is very high, the page table capacity is too large to fit on chip
- A translation lookaside buffer (TLB) caches the virtual to physical page number translation for recent accesses
- A TLB miss requires us to access the page table, which may not even be found in the cache – two expensive memory look-ups to access one word of data!
- A large page size can increase the coverage of the TLB and reduce the capacity of the page table, but also increases memory waste

TLB and Cache Access

TLB and Cache

- Is the cache indexed with virtual or physical address?
 - To index with a physical address, we will have to first look up the TLB, then the cache → longer access time
 - Multiple virtual addresses can map to the same physical address – must ensure that these different virtual addresses will map to the same location in cache – else, there will be two different copies of the same physical memory word
- Does the tag array store virtual or physical addresses?
 - Since multiple virtual addresses can map to the same physical address, a virtual tag comparison can flag a miss even if the correct physical memory word is present

Cache and TLB Pipeline



Virtually Indexed; Physically Tagged Cache

Bad Events

- Consider the longest latency possible for a load instruction:
 - TLB miss: must look up page table to find translation for v.page P
 - Calculate the virtual memory address for the page table entry that has the translation for page P – let's say, this is v.page Q
 - TLB miss for v.page Q: will require navigation of a hierarchical page table (let's ignore this case for now and assume we have succeeded in finding the physical memory location (R) for page Q)
 - Access memory location R (find this either in L1, L2, or memory)
 - We now have the translation for v.page P – put this into the TLB
 - We now have a TLB hit and know the physical page number – this allows us to do tag comparison and check the L1 cache for a hit
 - If there's a miss in L1, check L2 – if that misses, check in memory
 - At any point, if the page table entry claims that the page is on disk, flag a page fault – the OS then copies the page from disk to memory and the hardware resumes what it was doing before the page fault ... phew!