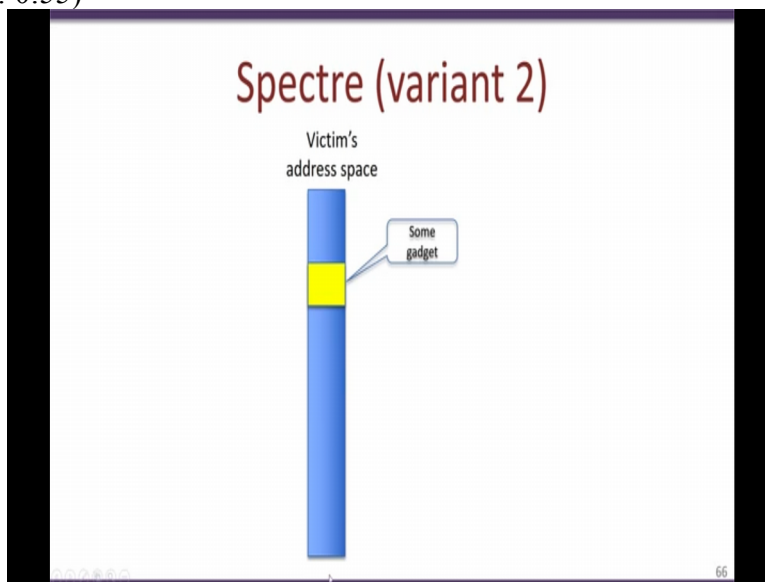


Information Security - 5 - Secure Systems Engineering
Professor Chester Rebeiro
Indian Institute of Technology, Madras
Spectre (Variant 2)

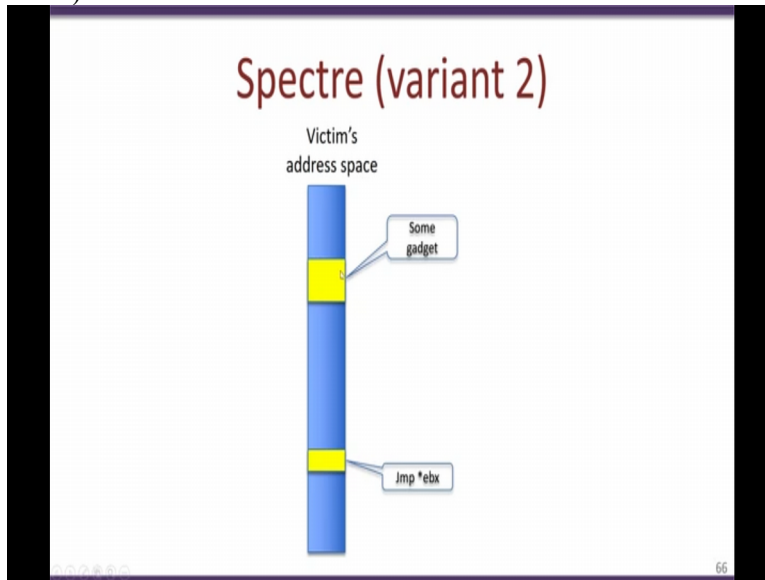
Hello and welcome to this lecture in the course for secure systems engineering so in the previous video lectures we had looked at the meltdown attack and also we looked at specter there's another variant of specter known as the variant 2 in this video we will look at this second variant of specter again as we have seen previously this variant is also based on the speculative execution of which is present in modern processors and it exploits this speculative execution in order to clean secret information out of another process.

(Refer Slide Time: 0:55)



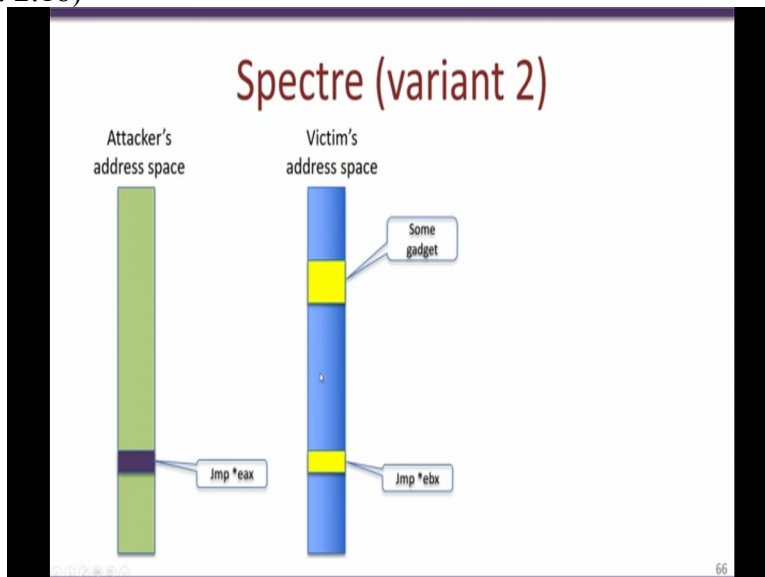
So the set up in the variant 2 is as follows we have a victim's address space so this is in an process address space off the victim and what we assume is that there is some portions of within this process space which has some secret data so what the attackers wants to do is that the attack a wants to actually obtain this secret data using the specter attack.

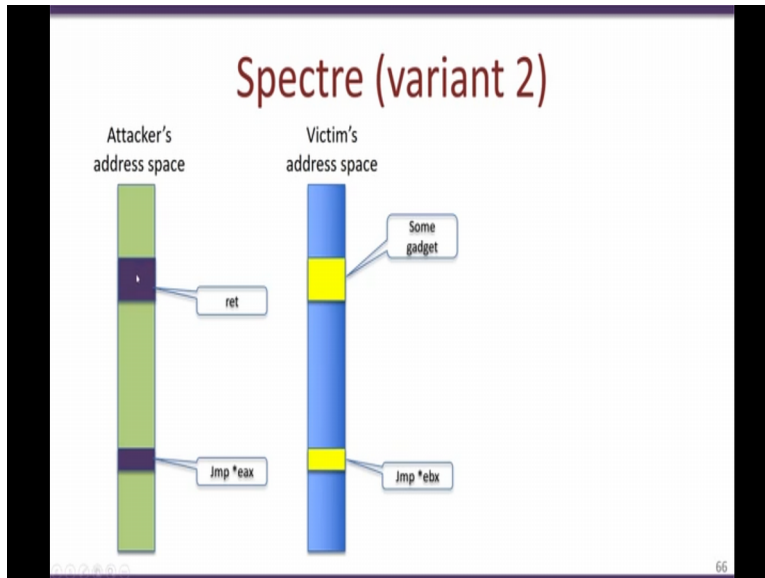
(Refer Slide Time: 1:32)



So the first thing that has done is that the attacker would identify 2 regions in the code so 1 it has an indirect jump based on a register value as shown over here and this jump is to some specific specter gadget which is present in the users victims user address space so this gadget did comprises of a few instructions that essentially would load into a register the contents of the secret information so what we see is that the attacker once you utilized this indirect Jump to this gadget and this gadget has instructions such as exhort or and something like that followed by a load instruction which includes secret data into a register.

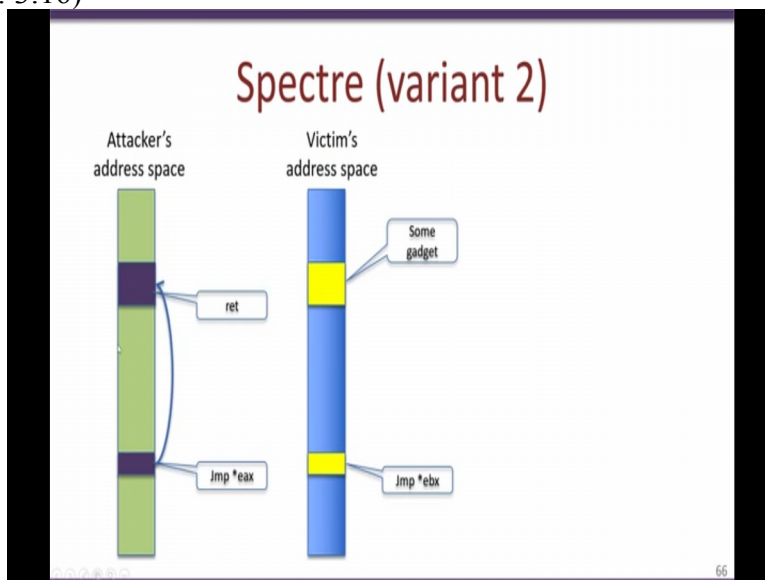
(Refer Slide Time: 2:18)





So in order to mount a spectre of variant 2 attack a what the attacker does is he creates his own attack program with a user space as follows so the attacker will first create an instruction over here which has some indirect branch so note that this address or the address of this particular location is exactly identical to the indirect branch present in the victim's address space so also what is assume is the attacker knows the address of some gadget and in his own address space replaces this area with a return so now what he does is that on the same processor that is executing this victim's process the attacker would run this attack program.

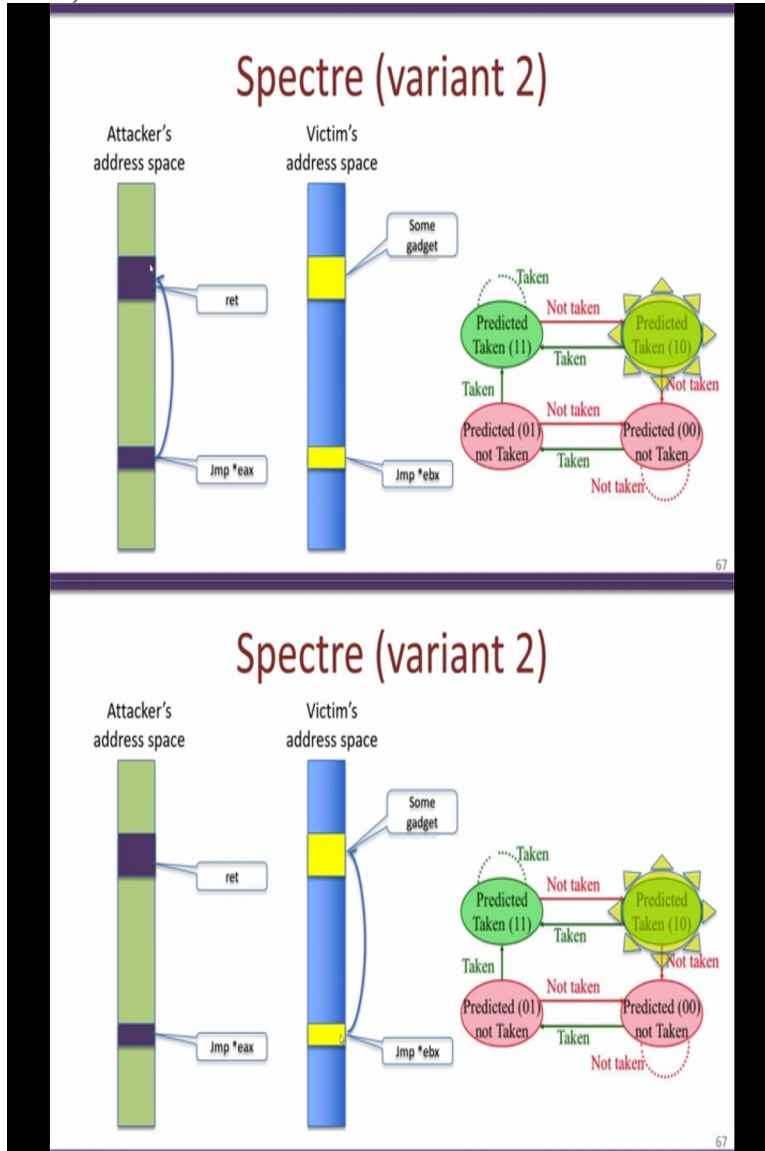
(Refer Slide Time: 3:16)



So therefore what would happen over here is repeatedly the attacker would make these indirect branches which Jump jumps to this region off memory and he keeps doing this in a loop

internally what happens is that the attackers is training the branch predictor that the next instruction following this indirect branch is the return instruction so the branch predictor based on its learning mechanisms like the thing like we have seen before would then be able to automatically start fetching data from this region and speculatively execute the instructions present in this region.

(Refer Slide Time: 3:57)

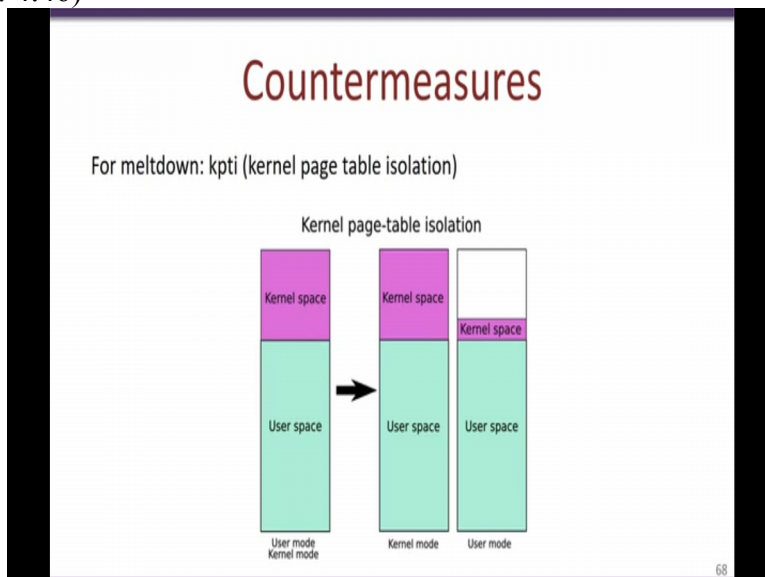


The wonder ability in this processor is that the branch predictor only looks at the virtual address in an address space and is not able to separate the virtual address of one process from and other process does when this branch in the victim also executes the branch predictor would look up its branch target buffer and it would find that the next address to be executed corresponds to this

address and it would start to speculatively execute this so recalled that this gadget which is present in the victim's address space is actually having some operations like add exhort or something followed by a lode operation which would speculatively load secret data from the victim's address space into a register.

And during this process as we've seen in the earlier videos the contents of the secret data would be present in the cache would modify the cache state and therefore techniques like the time required for a cache hit and cache miss can be used and therefore techniques like the time required for a cache hit and cache miss can be used to identify what the secret data is does what we have seen with here is that in this variant 2 the attacker using a completely isolated process is able to craft particular indirect branch in his own address space and clean the branch predictor in the processor to speculatively execute now since so this speculation would force secret data to be loaded into a register a which can then be use to retrieve information about the contents of the victims address space.

(Refer Slide Time: 4:46)

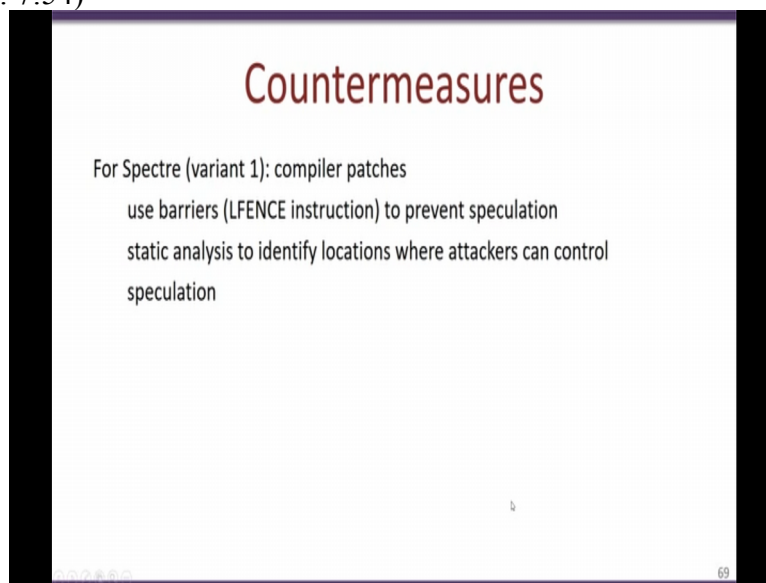


So in the last year there have been a lot of different countermeasures that have been developed for meltdown inspectors one of the important thing to actually prevent meltdown was based on restructuring the operating system address space so in the general practice prior to 2018 the user and kernel space was arranged in this particularly way so we had this as the virtual address space for a process it was divided into two regions a one was the user space up to a certain memory location and beyond that memory location was the kernel space so for example in a 32 bit Linux

system this of boundary was at the location zero X C followed by seven zeros below this location was a user space and above this location was the kernel space.

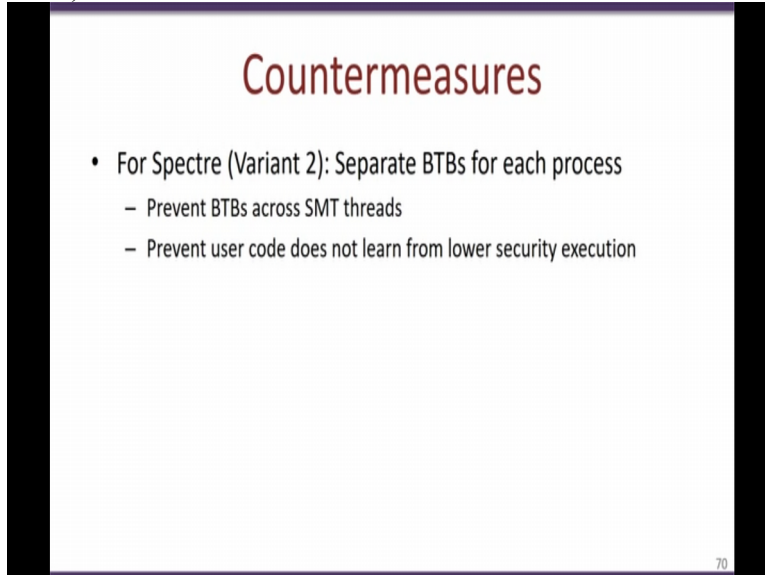
Now the melt down attack works because a user space program could exploit the speculative of behavior off the processor to be able to read contents of the kernel space the countermeasures work for this was known as KPTI or Kernel page table isolation in fact there were two sets of page tables one known as one which was activated in the was on user mode the other in the kernel mode so in the use of what the entire page tables that map to the kernel code was not present only a small stub for the kernel space was present so whenever the user space program invokes a system call it would be first trapped into this kernel space which would then shift more to the kernel mode and map the entire kernel space into the region similarly on return from the system call the virtual space would go back into this user mode now if a meltdown type of attack was actually utilized it has to be done from the user space and therefore would not be able to actually read any of the kernel space memory because the kernel space memory is not mapped in this particular mode.

(Refer Slide Time: 7:54)



Other techniques were suggested for the spectre first variant was compiler patches which uses barriers such as the LFENCE instruction to prevent speculation so the LFENCE instruction is supported by X 86 processors which is used actually sequence tallied execution so the LFENCE instruction would therefore prevent any speculation of beyond that instruction.

(Refer Slide Time: 8:20)



Countermeasures

- For Spectre (Variant 2): Separate BTBs for each process
 - Prevent BTBs across SMT threads
 - Prevent user code does not learn from lower security execution

70

In order to prevent variant 2 suggestions were need to have different branch target buffers present in the branch predictor unit each branch target buffer would cater to single process so for example if we had a processor with hyper 13 supported simultaneously to SMT threads then that would be two separate BTBs that are present so this would imply that the specter variant 2 will not work because each process or each thread which is running simultaneously would be using its own BTB and therefore the attacks where one prediction in one thread affecting speculative execution in another thread will not happen, thank you.