CS6570 Assignment 6 - Meltdown - Report

Rachit Tibrewal (CS16B022)
E Santhosh Kumar (CS16B107)

1. The first demo ./test chooses a character array in global data and fetches its contents using libkdump_read. It works as as expected. This indicates that libkdump_read is able to read from addresses in the address space of the current process. (refer Figure 1)

Figure 1: Demo 1 - First Test

2. The second demo ./kaslr fails to identify the offset of the direct physical map that maps the entire physical memory. Hence we use the kernel module in kaslr_offset to get the offset direcltly. The offset was found to be at 0xffff880000000000. (refer Figure 2)

Figure 2: Demo 2 - Breaking KASLR

3. For the third demo, we use the above found offset and run ./reliability with kalsr turned on. The success rate was observed to be 95.65% after trying to read around 2000 values (refer Figure 3).

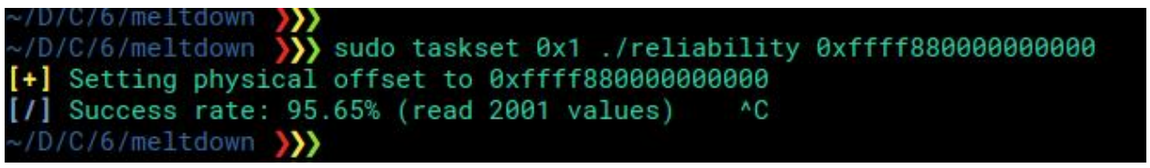

Figure 3: Demo 3 - reliability

4. For the fourth demo, we run a target process ./secret (refer Figure 4) that contains a secret string who's physical address is known. We then simultaneously run the attack process ./physical_reader (refer Figure 5) to read this string. We pass the physical address, along with direct physical map offset to the attacker, so that we know the final offset to start reading from. As we see in Figure 5, the attacker manages to read the string with some errors (the first word is read incorrectly).

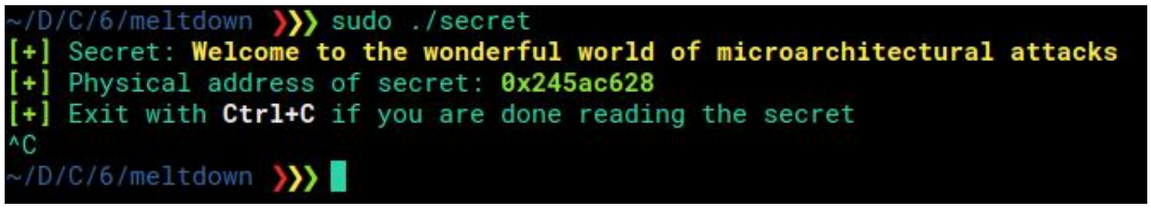

Figure 4: Demo 4 - Read physical memory - Target

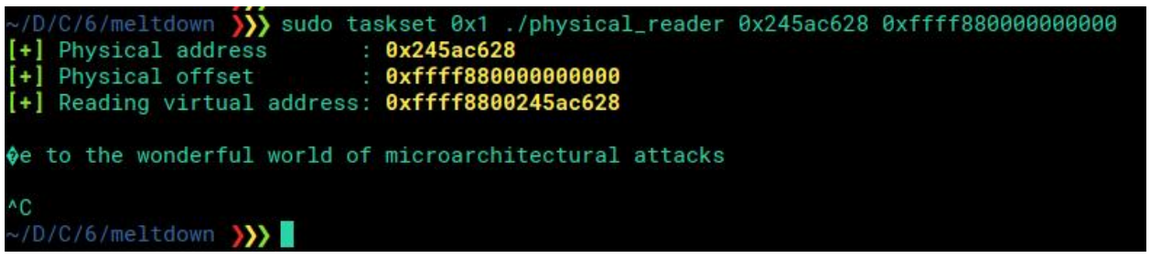

Figure 5: Demo 4 - Read physical memory - Attacker

5. In the fifth demo, we try dumping the physical memory starting from an offset. We first run a process ./memory_filler (refer Figure 6) to fill 4GB of memory with human readable character. We then simultaneously run the attack process ./memdump (refer Figure 7) to start dumping.

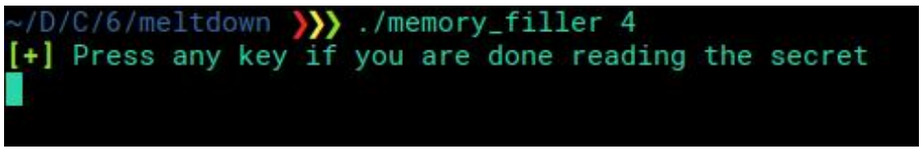

Figure 6: Demo 5 - Dump the memory - fills memory

Figure 7: Demo 5 - Dump the memory - Attacker