MULTICORE EX1

Yuval Katsman 323830448

As a disclaimer, given that I don't have much experience with operating system mechanisms, cybersecurity or in general with the many convoluted and abstraction breaking ideas that hardware presents.. I don't think that my hypothesis as to why the attack doesn't work is even close to being 'correct', and that there are many issues and oversights that would be apparent in my attempted implementation of it.

I tried reading and understanding the supplied spectre code to the best of my ability and then modified it to try and fit the proposed attack. The original spectre code successfully leaked the secret, while my modified version did not.

1 The Modifications

In my modified version I replaced the victim function with a simple **array1[x] = secret[x]** store. I think it is possible to leave the *if* statement intact, do the whole training section and force the speculation of the store, followed by supposedly load-forwarding the stored value to a controlled 4K aligned address within the speculation and moving that to a shared array like the normal spectre leak. But I didn't do it for the sake of simplifying the toy example.

```c
void victim_function(size_t x) {
    array1[x] = secret[x];
}
```

Given that I removed the *if* statement, I also removed the training section that seems to have both trained the branch predictor to go into the *if*, and at the same time delay the evaluation of the condition so that the side-channel creating instruction could be executed?

In the original spectre code the training was the part that called the victim function, so with it gone I added a call to the victim function on my own, followed by a load to a controlled 4K aligned address to the store in the victim function to **array3**, and then using that seemingly load-forwarded value I tried leaking it through a side-channel using a load to **array2**. This **array3** is a copy I made of the original **array1** that is also 4K aligned to it.

```
victim_function(malicious_x);               // attack!!!!!!
uint8_t fake_secret = array3[malicious_x]; // load - this gets squashed
temp &= array2[fake_secret * 512];          // squashed too, hopefully
```

The motivation behind this sequence of instructions was that assuming speculative load-forwarding happens, and hoping that the leaking instruction executes, one would expect the cache side-channel side-effect of the load to **array2** to take place, after which one could continue to execute the normal spectre code and attain it? Assuming they could deal with the squashing.

When the squashing happens and the two instructions are re-executed, **fake secret** would get a value between 1 and 16, depending on the **malicious x** used, and this specific multiple of 256 in **array2** would, similar to the original spectre code, not get counted as a cache-hit, in a sense making it not affect the histogram of possible values for the cache-hits.

2 Possible Ideas and Further Research

The attack did not work given my implementation. I'm not sure if I could assume based on it not working that Herbert's idea or his line of reasoning were flawed, but I would guess that what might have happened is similar to generally storing to an address based on a register's value, and loading to another address based on another register's value.

As far as I understood, if the store address is not ready and the load one is, instead of re-ordering these instructions and executing the load first the processor delays the load until the store is complete, or that the load address is ready. I think that it's possible that the load in this load-forward example is just delayed until the address is resolved as opposed to being speculatively forwarded the store value.

I would probably try to run some experiments with timing stores followed by loads of different address matches, given that the intel manual has observed delays with 4K aligned addresses, but hasn't really specified if they were addresses that had the same physical address or not. And also just to see for myself if Herbert's claims could be made more plausible; Then again I'm not sure if I would time just the load or both the load and store.

I would expect to see generally slower times with 4K aligned addresses that don't map to the same physical address if Herbert's claim was true, and times that are potentially as fast as having the same physical address for those that 4K align and match physical addresses given the 'right' speculation. If there was just a delayed load I would expect to see slower times in general for both types of 4K alignment.