Spectre and Meltdown Attack Variants

N V S S Hari Krishna Nama - 170050077
Dileep Kumar Reddy Chagam - 170050080
Tarun Somavarapu - 170050093

ABSTRACT

Modern processors use branch prediction and speculative execution to maximize performance. For example, if the destination of a branch depends on a memory value that is in the process of being read, CPUs will try to guess the destination and attempt to execute ahead. When the memory value finally arrives, the CPU either discards or commits the speculative computation. Speculative logic is unfaithful in how it executes, can access the victim's memory and registers, and can perform operations with measurable side effects.

Meltdown and Spectre work on personal computers, mobile devices, and in the cloud. Depending on the cloud provider's infrastructure, it might be possible to steal data from other customers. This makes the understanding of the basis of these exploits crucial for any computer scientist/engineer.

INTRODUCTION

Several microarchitectural design techniques have facilitated the increase in processor speed over the past decades. One such advancement is speculative execution, which is widely used to increase performance and involves having the CPU guess likely future execution directions and prematurely execute instructions on these paths. More specifically, consider an example where the program's control flow depends on an uncached value located in external physical memory. As this memory is much slower than the CPU, it often takes several hundred clock cycles before the value becomes known. Rather than wasting these cycles by idling, the CPU attempts to guess the direction of control flow, saves a checkpoint of its register state, and proceeds to speculatively execute the program on the guessed path. When the value eventually arrives from memory, the CPU checks the correctness of its initial guess. If the guess was wrong, the CPU discards the incorrect speculative execution by reverting the register state back to the stored checkpoint, resulting in performance comparable to idling. However, if the guess was correct, the speculative execution results are committed, yielding a significant performance gain as useful work was accomplished during the delay.

From a security perspective, speculative execution involves executing a program in possibly incorrect ways. However, because CPUs are designed to maintain functional correctness by reverting the results of incorrect speculative executions to their prior states, these errors were previously assumed to be safe.

By influencing which transient instructions are speculatively executed, we are able to leak information from within the victim's memory address space. In this report, we will explain two variants of Spectre and 1 variant of meltdown.

Variant 1: Exploiting Conditional Branches

In this variant of Spectre attacks, the attacker mistrains the CPU's branch predictor into mispredicting the direction of a branch, causing the CPU to temporarily violate program semantics by executing code that would not have been executed otherwise. As we show, this incorrect speculative execution allows an attacker to read secret information stored in the program's address space. Indeed, consider the following code example:

```
if (x < array1_size)
      y = array2[array1[x] * 4096];
```

Note that the read from array2 loads data into the cache at an address that is dependent on array1[x] using the malicious x. When the result of the bounds check is eventually determined, the CPU discovers its error and reverts any changes made to its nominal microarchitectural state. However, changes made to the cache state are not reverted, so the attacker can analyze the cache contents and find the value of the potentially secret byte retrieved in the out-of-bounds read from the victim's memory

Variant 2: Exploiting Indirect Branches

In this variant, the attacker chooses a gadget from the victim's address space and influences the victim to speculatively execute the gadget. The attacker does not rely on a vulnerability in the victim code. Instead, the attacker trains the Branch Target Buffer (BTB) to mispredict a branch from an indirect branch instruction to the address of the gadget, resulting in speculative execution of the gadget.

To mistrain the BTB, the attacker finds the virtual address of the gadget in the victim's address space, then performs indirect branches to this address. This training is done from the attacker's address space. It does not matter what resides at the gadget address in the attacker's address space; all that is required is that the attacker's virtual addresses during training match (or alias to) those of the victim. In fact, as long as the attacker handles exceptions, the attack can work even if there is no code mapped at the virtual address of the gadget in the attacker's address space

Meltdown:

The security of computer systems fundamentally relies on memory isolation, e.g., kernel address ranges are marked as non-accessible and are protected from user access. Out-of-order execution is an indispensable performance feature and present in a wide range of modern processors. Meltdown exploits side effects of out-of-order execution to read arbitrary kernel-memory locations including personal data and passwords.

- Meltdown exploits a race condition, which occurs between memory access and privilege checking during instruction processing
- Additionally, combined with a cache side-channel attack, this vulnerability allows a process to bypass the normal privilege checks that isolate the exploit process from accessing data belonging to the operating system and other running processes.
- Since many operating systems map physical memory, kernel processes, and other running user space processes into the address space of every process, Meltdown effectively makes it possible for a rogue process to read any physical, kernel, or other processes' mapped memory, regardless of whether it should be able to do so.

Meltdown Exploit overview:

- When the CPU attempts to execute an instruction referencing a forbidden memory address, the execution unit schedules both the privilege check and the memory access

- The privilege check informs the execution unit that the address involved in the access is forbidden to the process, and thus the instruction should fail.
- Because of the out-of-order execution, caching of data at the address which we are accessing may have been completed as a side effect of the memory access before the privilege check. The mere act of caching constitutes a leak of information in and of itself
- With use of timing attacks like Flush + Reload to get the information present in the cache.

BACKGROUND

Here, we describe some of the microarchitectural components of modern high-speed processors, how they improve performance, and how they can leak information from running programs.

A. Out-of-order Execution

Out-of-order execution is an optimization technique that allows maximizing the utilization of all execution units of a CPU core as exhaustive as possible. Instead of processing instructions strictly in the sequential program order, the CPU executes them as soon as all required resources are available

B. Speculative Execution

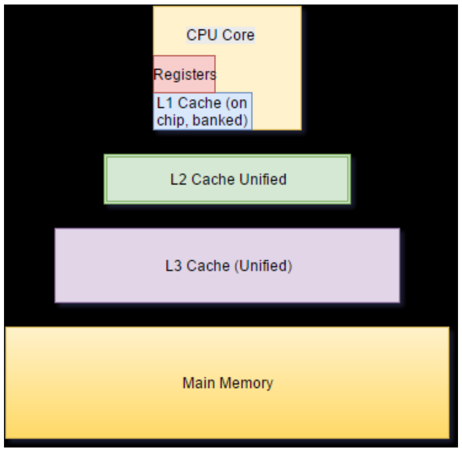
Often, the processor does not know the future instruction stream of a program. For example, this occurs when out-of-order execution reaches a conditional branch instruction whose direction depends on preceding instructions whose execution is not completed yet. In such cases, the processor can preserve its current register state, make a prediction as to the path that the program will follow, and speculatively execute instructions along the path. If the prediction turns out to be correct, the results of the speculative execution are committed (i.e., saved), yielding a performance advantage over idling during the wait. Otherwise, when the processor determines that it followed the wrong path, it abandons the work it performed speculatively by reverting its register state and resuming along the correct path

C. Branch Prediction

During speculative execution, the processor makes guesses as to the likely outcome of branch instructions. Better predictions improve performance by increasing the number of speculatively executed operations that can be successfully committed. The branch predictors of modern Intel processors, e.g., Haswell Xeon processors, have multiple prediction mechanisms for direct and indirect branches. Indirect branch instructions can jump to arbitrary target addresses computed at runtime. Consequently, several processor components are used for predicting the outcome of branches. The Branch Target Buffer (BTB) keeps a mapping from addresses of recently executed branch instructions to destination addresses. Processors can use the BTB to predict future code addresses even before decoding the branch instructions

D. The Memory Hierarchy

The memory hierarchy separates computer storage into a hierarchy based on response time.



E. Microarchitectural Side-Channel Attacks

When multiple programs execute on the same hardware, either concurrently or via time-sharing, changes in the microarchitectural state caused by the behavior of one program may affect other programs. This, in turn, may result in unintended information leaks from one program to another. In this project, we use the Flush+Reload technique, and its variant Evict+Reload, for leaking sensitive information. The main difference between the two techniques is the mechanism used for evicting the monitored cache line from the cache. In the Flush+Reload technique, the attacker uses a dedicated machine instruction, e.g., x86's clflush, to evict the line. Using Evict+Reload, eviction is achieved by forcing contention on the cache set that stores the line, e.g., by accessing other memory locations which are loaded into the cache and (due to the limited size of the cache) cause the processor to discard (evict) the line that is subsequently probed

F. Return-Oriented Programming

Return-Oriented Programming (ROP) is a technique that allows an attacker who hijacks control flow to make a victim perform complex operations by chaining together machine code snippets, called gadgets, found in the code of the vulnerable victim

ATTACK OVERVIEW

VARIANT 1: EXPLOITING CONDITIONAL BRANCH MISPREDICTION
VARIANT 2: POISONING INDIRECT BRANCHES
MELTDOWN: READING KERNEL MEMORY FROM USER SPACE

Differences between Spectre and Meltdown

1. Meltdown does not use branch prediction. Instead, it relies on the observation that when an instruction causes a trap, following instructions are executed out-of-order before being terminated
2. Meltdown exploits a vulnerability specific to many Intel and some ARM processors

PERSONAL SECTION

Spectre:

When we executed the example code accompanying the original white paper demonstrating the reading of memory using a Spectre attack on x86, we were skeptical about it working on our machines due to many patches already in action. The simple attack worked on some older machines, but it couldn't work on a little modern laptop. After going through many implementations, we found that the original paper's ATTACK PATTERN is critical. The stride prediction by the system is preventing us from implementing the timing-based cache attack. We quickly randomized the attack pattern and increased the number of training loops for misdirecting the branch prediction. And now it worked seamlessly.