Manual Of
The Microarchitectural Attack

Hardware Security

For questions contact:
Mathé @ m.c.hertogh@vu.nl
or
Max @ m.barger2@vu.nl

Project Goal:

The goal of this project is to understand in-depth how internal CPU components work and how certain CPU microarchitectural design decisions can lead these components to leak sensitive information through the side-channels which an attacker can observe and exploit to compromise current modern computer systems.

Overview:

Week 1:

In the first week, you will study two CPU buffers:

- The `Line Fill Buffer (LFB)`, which is an optimization implemented on the microarchitectural level of each physical core (i.e. used by the two hyperthreads of each core), to speculatively serve load operations with data, without the notion of virtual addresses and therefore without the awareness of the different address spaces, access privileges, or security domains, and this is what the **Rogue In-Flight Data Load (RIDL)** attack exploits to leak sensitive information by abusing transient execution breaking the different security boundaries.

- The `Staging Buffer`, is a microarchitectural optimization implemented outside of the CPU cores (i.e. shared by all hyperthreads of all physical cores). Some complex microcoded instructions initiate offcore requests to certain CPU components located outside the CPU cores, like for example the `Power Control Unit (PCU)` and the `Digital Random Number Generator (DRNG)`, which serve all CPU cores. The staging buffer hosts the responses coming from the offcore components, and later the microcode of those instructions which initiated the offcore requests will fetch the corresponding offcore responses from the staging buffer … but by design, those instructions will fetch from that buffer more than the responses they asked for and this is what the **CrossTalk** attack exploits to leak other sensitive information located in the staging buffer and directed to other CPU cores.

Week 2:

In the second week of the project, you will study a type of Machine Clear (or pipeline flush) based on the floating-point unit (FPU) and how it behaves when operating on denormal/subnormal numbers, which are a special range of floating-point numbers with a value smaller than the smallest Normal number (i.e., 2^-1022).
The hardware implementation of modern FPUs is not designed to handle denormal numbers, and always speculates to be operating on normal floating-point numbers instead, which is what it is optimized for. So when either the operands or the result of a floating-point operation happen to be a denormal number, the FPU doesn't detect this immediately, creating a transient window where it computes a wrong result first (i.e., a transient result), forwarding it to subsequent instructions, thus injecting a wrong result in all the instructions dependent on this operation. When later the FPU detects the denormal number, it warns the CPU, which now has to flush its pipeline (i.e., issue a Machine Clear), issue a microcode assist to re-perform the floating-point operation representing correctly the detected denormal number, and finally reforward the correct result (i.e., the architectural result) to all subsequent dependent operations.

Week 3:

In the third week, you will combine all three techniques (RIDL, CrossTalk, and FP Machine Clear) to perform a local privilege escalation attack on the cluster provided.

You will find a privileged program on the cluster that does the following:

1. Request 2 random numbers from the DRNG.
2. Use the obtained numbers as operands of a floating-point division, and use the transient result of this division as the prefix string of the root password.
3. Concatenate the computed prefix to a few fixed characters.
4. Set the resulting string as the new root password, updating its corresponding hash in `/etc/shadow` as usual.

First, you will have to leak across physical cores the requested random numbers, which will be temporarily stored in the staging buffer by the DRNG, so by performing the CrossTalk attack, you should be able to leak them while the privileged program is reading them from the staging buffer (which is shared between all cores).
Second, once you have the random numbers, it means that you have the operands of the floating-point division that the privileged program does internally, so you can replicate the same division and leak the corresponding transient result, thus obtaining the root password's prefix string.
Third, you will leak across hyperthreads the new password's hash stored in the `/etc/shadow` file by performing the RIDL attack.
Finally, by knowing both the password's prefix string and hash, you will be able to crack the remaining fixed characters of the root password and gain control over the target machine.

Background:

1) Read carefully the RIDL and CrossTalk papers for week 1, and the Machine Clear paper for week 2 and fully understand them.
2) Understand the different cache levels and their corresponding properties, granularity, set-associativity, inclusiveness, etc. You must have a clear understanding of how to flush specific addresses from the cache, which is the base for the `Flush+Reload` attack used in all the attacks you will implement to leak the secret information.
3) Out-of-Order Execution engine with its different stages, i.e., In-Order register allocation and renaming, Out-of-Order scheduling and execution of `micro-operations (uops)`, Retirement unit managing the completed uops.
4) Speculative execution. When does it occur? What happens to the pipeline in case of a misprediction?
5) In-Flight data, the different types of data that can be leaked from internal `CPU` components during speculative execution. `RIDL` focuses on the `LFB`, `Store Buffer (SB)`, and `Load Ports (LP)`.
6) The staging buffer, offcore requests, responses, and the different types and offsets.
7) Understand how to bring to a core-local `LFB` the secret information requested from another physical core.
8) Understand how to distinguish the secret information you're interested in leaking from the noise present in the LFB.
9) Understand what is a machine clear, the different types and how it can be observed through a side-channel.
10) How FPUs handle denormal numbers, how to leak the transient results of denormal floating-point operations.

(Inline) Assembly

Contrary to the labs, you are free to use (inline) assembly in this big project, alongside your C code.