SECURITY ANALYSIS OF AMD PREDICTIVE STORE FORWARDING

March 2021

INTRODUCTION

AMD "Zen3" processors feature a new technology called Predictive Store Forwarding (PSF). PSF is a hardware-based micro-architectural optimization designed to improve the performance of code execution by predicting dependencies between loads and stores. Like technologies such as branch prediction, with PSF the processor "guesses" what the result of a load is likely to be, and speculatively executes subsequent instructions. In the event that the processor incorrectly speculated on the result of the load, it is designed to detect this and flush the incorrect results from the CPU pipeline.

Security research in recent years has examined the security implications of incorrect CPU speculation and how in some cases it may lead to side channel attacks. For instance, conditional branch speculation, indirect branch speculation, and store bypass speculation have been demonstrated to have the potential to be used in side-channel attacks (*e.g.*, Spectre v1, v2, and v4 respectively).

This whitepaper describes the PSF feature, how it works, and associated potential security concerns. The paper also includes information about hardware controls for disabling the feature when needed for security reasons, and preliminary information about proposed Linux interfaces for those controls.

PREDICTIVE STORE FORWARDING

It is common for a CPU to execute a load instruction to an address that was recently written by a store. Many modern processors implement a technique known as Store-To-Load-Forwarding (STLF) to improve performance in such cases. With STLF, data from the store is forwarded directly to the load without having to wait for it to be written to memory. In a typical CPU, STLF occurs after the address of both the load and store are calculated and determined to match.

PSF expands on this by speculating on the relationship between loads and stores without waiting for the address calculation to complete. With PSF, the CPU learns over time the relationship between loads and stores. If STLF typically occurs between a particular store and load, the CPU will remember this. When the CPU sees the store/load pair again, it may predict that STLF will occur and speculatively forward the data from the store to the load. This is done before confirming that the store and load are in fact to the same address.

In typical code, PSF provides a performance benefit by speculating on the load result and allowing later instructions to begin execution sooner than they otherwise would be able to. Most of the time, the PSF prediction is accurate. However, there are cases where the prediction may not be accurate and cause incorrect CPU speculation.

CAUSES OF INCORRECT PSF

Incorrect PSF predictions can occur due to at least the following two reasons. First, it is possible that the store/load pair had a dependency for a while but later stops having a dependency. This can occur if the address of either the store or load changes during the execution of the program.

The second source of incorrect PSF predictions can occur if there is an alias in the PSF predictor structure. The PSF predictor is designed to track stores/load pairs based on portions of their RIP. It is possible that a store/load pair which does have a dependency may alias in the predictor with another store/load pair which does not. This may result in incorrect speculation when the second store/load pair is executed.

EXAMPLE

Consider the following exemplary C code and corresponding assembly.

```
void fn(int idx) {
        unsigned char v;
        idx_array[0] = 4096;
        v = array[idx_array[idx] * (idx)];
}
```

```
0000000000000a60 <fn>:
 a60:   48 8d 05 99 55 20 00    lea    0x205599(%rip),%rax        # 206000 <idx_array>
 a67:   48 63 d7                movslq %edi,%rdx
 a6a:   c7 05 8c 55 20 00 00    movl   $0x1000,0x20558c(%rip)        # 206000 <idx_array>
 a71:   10 00 00
 a74:   8b 04 90                mov    (%rax,%rdx,4),%eax
 a77:   0f af f8                imul   %eax,%edi
 a7a:   48 8d 05 7f 25 20 00    lea    0x20257f(%rip),%rax        # 203000 <array>
 a81:   0f b6 04 38             movzbl (%rax,%rdi,1),%eax
```

This function takes a parameter (idx) which in our example will either be 0 or 1. After setting idx_array[0] the code then accesses idx_array[idx]. In this program, assume idx_array[] is initially all 0's. Therefore, when this function is called, it always results in accessing array[0]. This is because if idx=0 the function accesses array[idx_array[0] * 0] = array[4096 * 0] = array[0], while if idx=1 the function accesses array[idx_array[1] * 1] = array[0 * 1] = array[0].

In the assembly code, there is a store at offset 0xa6a and a load at offset 0xa74. If idx=0 then this store and load are to the same address and STLF will occur. However, if idx=1 then they are no longer to the same address.

For AMD Zen3 processors with PSF, it is possible that the CPU could incorrectly predict that the store at offset 0xa6a will forward to the load at 0xa74 even if idx=1. This may occur if the function was called many times with idx=0 first, and then later called with idx=1. If this incorrect prediction occurs, the CPU will speculatively forward the store data (4096) to the load at offset 0xa74. The CPU will then multiply this value by idx (1) and attempt a load of array[4096].

Note that architecturally, array[4096] is not accessed, but due to PSF, a speculative access to this index may occur. This may be detected by later timing an access to this array index.

This simple example demonstrates how incorrect PSF speculation may occur in a detectable manner.

LIMITATIONS

There are a number of limitations to the PSF feature and the speculation that may occur as a result of incorrect PSF predictions.

TRAINING LIMITATIONS

The PSF is limited to training about store/load dependencies within the same context. A context is defined by the current values of CPL, ASID, PCID, CR3, and SMM status. Training only occurs if both the store and load execute in the same context. Any time that any piece of the context state changes (e.g. system call) existing training information is flushed. In particular, this flushing occurs on all far control transfers which includes all CPL changes, system call and return, interrupt/exceptions, SMM entry/exit, and VM entry/exit.

Note that the PSF predictor is partitioned amongst SMT threads so the activity of one SMT thread does not influence the PSF predictions of the sibling thread.

Finally, the store and load used to train the PSF must be relatively close together in the instruction stream and there cannot be any pipeline flushes (such as due to a mis-predicted branch) between the store and the load.

Speculation Limitations

As with the CPU speculation that occurs from branch mispredictions, speculation due to bad PSF predictions occurs within the current process context. Speculative memory accesses performed are subject to standard paging checks and only memory accessible in the current context and at the current privilege level may be speculatively accessed. Speculation ceases if the processor encounters certain types of serializing operations such as LFENCE.

Security Analysis

Previous research has shown that when CPUs speculate on non-architectural paths it can lead to the potential of side channel attacks. In particular, programs that implement isolation, also known as 'sandboxing', entirely in software may need to be concerned with incorrect CPU speculation, which can occur due to bad PSF predictions.

Because PSF speculation is limited to the current program context, the impact of bad PSF speculation is similar to that of speculative store bypass (*e.g.*, Spectre v4). In both cases, a security concern arises if code exists that implements some kind of security control which can be bypassed when the CPU speculates incorrectly. This may occur if a program (such as a web browser) hosts pieces of untrusted code and the untrusted code is able to influence how the CPU speculates in other regions in a way that results in data leakage. This is similar to the security risk with other Spectre-type attacks.

It is important to understand that because aliases may occur within the PSF structure, incorrect predictions may occur on loads even if the load is not directly controllable by an attacker. If an attacker is able to run code within a target application, they may be able to influence speculation on other loads within the same application by purposely training the PSF predictor with malicious information.

As an example, consider a software sandbox that implements protection for Spectre v1 by masking an array index before using it. An attacker may be able to construct a PSF alias so that the processor predicts the load of the array mask to be an incorrect value, thus bypassing the Spectre v1 protection mechanism.

On the other hand, isolation that is done using hardware mechanisms, such as separate address spaces, may be considered safe from Spectre-style attacks including bad PSF speculation since PSF speculation cannot occur across address spaces. It is important to note that PSF training also cannot occur across privilege domains since the predictor structure is flushed on such changes. For example, user space code execution cannot influence PSF predictions done in the kernel.

Customers with software that implements sandboxing and are concerned about the PSF behavior on AMD Zen3 processors may choose to disable the PSF functionality as described in the following section.

Predictive Store Forwarding Controls

MSR Controls

There are two hardware control bits for the PSF feature:

- MSR 48h bit 2 – Speculative Store Bypass (SSBD)
- MSR 48h bit 7 – Predictive Store Forwarding Disable (PSFD) (NEW in Zen3)

The PSF feature is disabled if either of these bits are set. These bits are controllable on a per-thread basis in an SMT system. By default, both SSBD and PSFD are 0 meaning that the speculation features are enabled.

While the SSBD bit disables PSF and speculative store bypass, PSFD only disables PSF. PSFD may be desirable for software which is concerned with the speculative behavior of PSF but desires a smaller performance impact than setting SSBD.

Support for PSFD is indicated in CPUID Fn8000_0008 EBX[28]. AMD processors that support PSF will also support PSFD.

Linux Controls

AMD has recently proposed Linux patches that enable control of the PSFD bit in MSR 48h. These patches implement the following behavior**:**

Kernel Command Line Parameter	**Behavior**
mitigations	If ‘off’, PSFD is set to 0. If ‘auto’, PSFD is also set to 0 (same as SSBD)
nopsfd	Sets PSFD to 0
psfd	If ‘on’ PSFD is set to 1. If ‘off’ PSFD is set to 0

Note that software that already uses pr_ctl to disable the PR_SPEC_STORE_BYPASS feature will be run with SSBD=1 which effectively disables PSF.

Conclusion

Predictive Store Forwarding is a new feature in AMD Zen3 CPUs which may improve application performance but also has security implications. While AMD is not currently aware of any code that would be considered vulnerable due to PSF behavior, this whitepaper examines the potential security implications of PSF, in general, as well as mechanisms that are designed to disable the feature if desired. AMD believes that for most applications, the security risk of PSF is likely low and where isolation is required, techniques such as address space isolation are preferred over software sandboxing.

AMD recommends leaving the Predictive Store Forwarding feature enabled as the default setting.