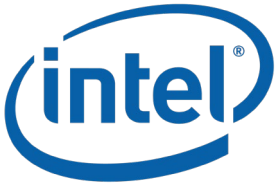

Intel Analysis of Speculative Execution Side Channels

White Paper

Revision 4.0

July 2018

Any future revisions to this content can be found at https://software.intel.com/security-software-guidance/software-guidance when new information is available. This archival document is no longer being updated.

Document Number: 336983-004

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. Check with your system manufacturer or retailer or learn more at www.intel.com.

Intel provides these materials "as is", with no express or implied warranties.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.

Contents

§

Revision History

Document Number	Revision Number	Description	Date
336983-001	1.0	Initial release.	January 2018
336983-002	2.0	Link to security research findings page on intel.com added to Section 1 "Introduction". Minor correction to Section 2.2 "Side Channel Cache Methods". Link to *Speculative Execution Side Channel Mitigations* added to Section 3 "Mitigations". Footnote with link to retpoline details added to Section 3.2 "Branch Target Injection Mitigation".	January 2018
336983-003	3.0	Added information on Variant 3a, Rogue System Register Read, and Variant 4, Speculative Store Bypass.	May 2018
336983-004	4.0	Added information on Bounds Check Bypass Store.	July 2018

§

1 Introduction

Intel is committed to improving the overall security of computer systems through hardware and software. As detailed by Google Project Zero, https://googleprojectzero.blogspot.com/ and various security researchers, a new series of side-channel analysis methods have been discovered that potentially facilitate access to unauthorized information. These methods rely on common properties of both high-performance microprocessors and modern operating systems. Susceptibility to these methods is not limited to Intel processors, nor does it imply the processor is working outside its intended functional specification. All of the methods take advantage of *speculative execution*, a common technique in processors used to achieve high performance.

Intel is working closely with our ecosystem partners, as well as with other silicon vendors whose processors are affected, to design mitigations for these methods.

This white paper provides information on various side channel methods, as well as the mitigations that Intel is pursuing for each.

For information and links to useful resources, visit the security research findings page on intel.com: https://www.intel.com/content/www/us/en/architecture-and-technology/facts-about-side-channel-analysis-and-intel-products.html.

2 Speculative Execution Side Channel Methods

2.1 Speculative Execution

Speculative execution is one of the main techniques used by most modern high performance processors to improve performance. The concept behind speculative execution is that instructions are executed ahead of knowing that they are required. Without speculative execution, the processor would need to wait for prior instructions to be resolved before executing subsequent ones. By executing instructions speculatively, performance can be increased by minimizing latency and extracting greater parallelism. The results may be discarded if it is discovered that the instructions were not needed after all.

The most common form of speculative execution involves the control flow of a program. Instead of waiting for all branch instructions to resolve to determine which operations are needed to execute, the processor predicts the control flow using a highly sophisticated set of mechanisms. Usually the predictions are correct, which allows high performance to be achieved by hiding the latency of the operations that determine the control flow and increasing the parallelism the processor can extract by having a larger pool of instructions to analyze. However, if a prediction is wrong, then the work that was executed speculatively is discarded and the processor will be redirected to execute down the correct instruction path.

While speculative operations do not affect the architectural state of the processor, they can affect the microarchitectural state, such as information stored in Translation Lookaside Buffers (TLBs) and caches. The side channel methods described in this white paper take advantage of the fact that the content of caches can be affected by speculative execution.

2.2 Side Channel Cache Methods

A side channel method works by gaining information through observing the system, such as by measuring microarchitectural properties about the system. Unlike buffer overflows and other vulnerability classes, side channels do not directly influence the execution of the program, nor do they allow data to be modified or deleted.

A cache timing side channel involves an agent detecting whether a piece of data is present in a specific level of the processor’s caches, where its presence may be used to infer some other piece of information. One method to detect whether the data in question is present is to use timers to measure the latency to access memory at the address. If the memory access takes a short time, then the data must be present in a nearby cache. If the access takes a longer time, then the data may not be in the nearby cache.

Several security researchers and teams have identified several methods through which a cache timing side channel can potentially be used to leak secret information.

2.2.1 Variant 1: Bounds Check Bypass

The bounds check bypass method takes advantage of speculative execution after conditional branch instructions. A malicious actor discovers or causes the creation of "confused deputy" code which allows the attacker to use speculative operations to infer information not normally accessible to the attacker.

This method uses speculative operations that occur while the processor is checking whether an input is in bounds, such as checking if the index of an array being read is within acceptable values. It takes advantage of memory accesses to out of bound memory that are performed speculatively before the bounds check resolves. These memory accesses can be used in certain circumstances to leak information to the attacker.

If the attacker can identify an appropriate "confused deputy" in a more privileged level, the malicious actor may be able to exploit that deputy to deduce the contents of memory accessible to that deputy but not to the malicious actor.

One subvariant of this technique, known as *bounds check bypass store*, is to use speculative stores to overwrite younger speculative loads in a way that creates a side channel controlled by a malicious actor.

Refer to the example bounds check bypass store sequence below:

```
int function(unsigned bound, unsigned long user_key) {
        unsigned long data[8], i;


        ;; bound is trusted and is never more than 8
        i = 0;
        while (i < bound) {
              data[i] = user_key;
              i++;
        }


        return 0;
        }
```

It is possible that the above sequence might speculatively overwrite the return address on the stack with `user_key`. This may allow a malicious actor to specify a `user_key` that is actually the instruction pointer of a disclosure gadget that they wish to be speculatively executed.

The steps below describe how an example attack using this method might occur:

1. The CPU conditional branch predictor predicts that the loop will iterate 10 iterations, when in reality the loop should have only executed 8 times. After the 10th iteration, the predictor will resolve, fall through, and execute the following instructions. However, the 9th iteration of the loop may speculatively overwrite the return address on the stack.

2. The CPU decodes the `RET` and speculatively fetches instructions based on the prediction in the return stack buffer (RSB). The CPU may speculatively execute those instructions.

3. `RET` loads the value that it believes is at the top of the stack (but which came from the speculative store of `user_key` in step 1) and redirects the instruction pointer to that value. The results of any operations speculatively executed in step 2 are discarded.

4. The disclosure gadget at the instruction pointer of `user_key` (which was specified by the malicious actor) speculatively executes and creates a side channel that can be used to reveal data specified by the malicious actor.

5. The conditional jump that should have ended the loop then executes and redirects the instruction pointer to the next instruction after the loop. This discards the speculative store of `user_key` that overwrote the return address on the stack, as well as all other operations between step 1 and step 4.

6. The CPU executes the `RET` again, and the program continues.

SMEP will prevent this attack from causing a supervisor `RET` to speculatively execute code in user mode page. Control flow Enforcement Technology (CET) can also help prevent speculative execution of instructions at incorrect indirect branch targets. For further information, refer to the *Bounds check bypass store attacks* section of the *Analyzing potential bounds check bypass vulnerabilities* white paper.

This example can be mitigated either by applying `LFENCE` before the `RET` (after the loop ends), by using bounds clipping to ensure that store operations do not occur outside of the array's bounds, even speculatively, or by ensuring that the incorrect return pointer is detected and that the return does not speculatively use the incorrect value.

2.2.2 Variant 2: Branch Target Injection

The *branch target injection* method takes advantage of the indirect branch predictors inside the processor that are used to direct what operations are speculatively executed. By influencing how the indirect branch predictors operate, an attacker can cause malicious code to be speculatively executed and then use the effects such code has on the caches to infer data values.

For conditional direct branches, there are only two options as to what code speculatively executes – either the target of the branch or the fall-through path of instructions directly subsequent to the branch. The attacker cannot cause code to be speculatively executed outside of those locations. Indirect branches, however, can cause speculative execution of code at a wider set of targets. This method works by causing an indirect branch to speculatively execute a 'gadget' which creates a side channel based on sensitive data available to the victim.

The ability to interfere with the processor's predictors to cause such a side channel is highly dependent on the microarchitectural implementation, and the exact methods used may thus vary across different processor families and generations. For example, the indirect branch predictors in some processor implementations may only use a subset of the overall address to index into the predictor. If an attacker can discern what subset of bits are used, the attacker can use this information to create interference due to aliasing. Similarly, on processors that support Intel® Hyper-Threading Technology (Intel® HT Technology), whether one thread's behavior can influence the prediction of the other thread is a consideration. The branch target injection method can only occur for a near indirect branch instruction.

2.2.3 Variant 3: Rogue Data Cache Load

The *rogue data cache load* method involves an application (user) attacker directly probing kernel (supervisor) memory. Such an operation would typically result in a program error (page fault due to page table permissions). However, it is possible for such an operation to be speculatively executed under certain conditions for certain implementations. For instance, on some implementations such a speculative operation will only pass data on to subsequent operations if the data is resident in the lowest level data cache (L1). This can allow the data in question to be queried by the application, leading to a side channel that reveals supervisor data. This method only applies to regions of memory designated supervisor-only by the page tables; not memory designated as not present.

2.2.4 Variant 3a: Rogue System Register Read

The *rogue system register read* method, as described as Variant 3a in the ARM* whitepaper[1], uses both speculative execution and side channel cache methods to infer the value of some processor system register state which is not architecturally accessible by the attacker. This method uses speculative execution of instructions that read system register state while the processor is operating at a mode or privilege level that does not architecturally allow the reading of that state. The set of system registers that can have their value inferred by this method is implementation-specific.

Although these operations will architecturally fault or VM exit, in certain cases, they may return data accessible to subsequent instructions in the speculative execution path. These subsequent instructions can then create a side channel to infer the system register state.

Intel's analysis is that the majority of state exposed by the Variant 3a method is not secret or sensitive, nor directly enables attack or exposure of user data. The use of the Variant 3a method by an attacker may result in the exposure of the physical addresses for some data structures and may also expose the linear addresses of some kernel software entry points.

Knowledge of these physical and linear addresses may enable attackers to determine the addresses of other kernel data and code elements, which may impact the efficacy of the Kernel Address Space Layout Randomization (KASLR) technique.

KASLR, as a security defense in-depth feature, has been subject to a number of attacks in recent years; in particular against local attackers who can control code execution. As the rogue system register read method involves attacker controlled code execution, a local attacker employing rogue system register read to break KASLR may be low impact for most end users.

2.2.5 Variant 4: Speculative Store Bypass

The *speculative store bypass* method takes advantage of a performance feature present in many high-performance processors that allows loads to speculatively execute even if the address of preceding potentially overlapping store is unknown. In such a case, this may allow a load to speculatively read a stale data value. The processor will eventually correct such cases, but an attacker may be able to discover "confused deputy" code which may allow them to use speculative execution to reveal the value of memory that is not normally accessible to them. In a language based security environment (e.g., a managed runtime), where an attacker is able to influence the generation of code, an attacker may be able to create such a confused deputy. Intel has not currently observed this method in

[1] ARM* whitepaper, "Cache Speculation Side-Channels" located here: https://armkeil.blob.core.windows.net/developer/Files/pdf/Cache_Speculation_Side-channels_03May18.pdf, published in May 2018

situations where the attacker has to discover such a confused deputy instead of being able to cause it to be generated.

3 Mitigations

Intel has been working closely with the ecosystem, including other processor vendors and software developers, to identify mitigations for the side channel methods previously described. The mitigation strategy is focused on identifying techniques that can be applicable both for products currently in the market as well as for future products in development. Mitigations pursued address the attack method in question, as well as balancing that with other considerations such as performance impact and complexity of implementation. Enabling existing processor security features like Supervisor-Mode Execution Protection and Execute Disable Bit can substantially increase the difficulty of attacking a system. See the Related Intel Security Features section for additional details on these security features. Intel strongly recommends following good security practices that protect against malware in general, including always staying updated with the latest software patches and microcode and not installing untrusted software.

Intel has been working with OS vendors, Virtual Machine Monitor Vendors, and other software developers to mitigate these attacks.

For additional details on mitigations, see *Speculative Execution Side Channel Mitigations*, located here: https://software.intel.com/sites/default/files/managed/c5/63/336996-Speculative-Execution-Side-Channel-Mitigations.pdf.

As part of our normal development process, Intel may enhance the efficacy of these mitigations in upcoming processors.

3.1 Variant 1: Bounds Check Bypass Mitigation

For the bounds check bypass method, Intel's mitigation strategy is focused on software modifications.

The software mitigation that Intel recommends is to insert a barrier to stop speculation in appropriate places. In particular, the use of an `LFENCE` instruction is recommended for this purpose. Serializing instructions, as well as the `LFENCE` instruction, will stop younger instructions from executing, even speculatively, before older instructions have retired but `LFENCE` is a better performance solution than other serializing instructions. An `LFENCE` instruction inserted after a bounds check will prevent younger operations from executing before the bound check retires. Note that the insertion of `LFENCE` must be done judiciously; if it is used too liberally, performance may be significantly compromised.

It is possible to create a set of static analysis rules in order to help find locations in software where a speculation barrier might be needed. Intel's analysis of the Linux kernel for example has only found a handful of places where `LFENCE` insertion is required, resulting in minimal performance impact. As with all static analysis tools, there are likely to be false positives in the results and human inspection is recommended.

3.2 Variant 2: Branch Target Injection Mitigation

For the branch target injection method, two mitigation techniques have been developed. This allows a software ecosystem to select the approach that works for particular security, performance and compatibility goals.

The first technique introduces a new interface between the processor and system software. This interface provides mechanisms that allow system software to prevent an attacker from controlling the victim's indirect branch predictions, such as flushing the indirect branch predictors at the appropriate time to mitigate such attacks. The details of this interface will be provided in a future revision of the Intel® 64 and IA-32 Architectures Software Developer's Manuals. This mitigation strategy requires both updated system software as well as a microcode update to be loaded to support the new interface for many existing processors. This new interface will also be supported on future Intel processors. There are three new capabilities that will now be supported for this mitigation strategy. These capabilities will be available on modern existing products if the appropriate microcode update is applied, as well as on future products, where the performance cost of these mitigations will be improved. In particular, the capabilities are:

- Indirect Branch Restricted Speculation (IBRS): Restricts speculation of indirect branches.
- Single Thread Indirect Branch Predictors (STIBP): Prevents indirect branch predictions from being controlled by the sibling Hyperthread.
- Indirect Branch Predictor Barrier (IBPB): Ensures that earlier code's behavior does not control later indirect branch predictions.

The second technique, developed by Google, introduces the concept of a "return trampoline", also known as "retpoline"[2]. Essentially, software replaces indirect near jump and call instructions with a code sequence that includes pushing the target of the branch in question onto the stack and then executing a `return` (`RET`) instruction to jump to that location, as `return` instructions can generally be protected using this method. This technique may perform better than the first technique for certain workloads on many current Intel processors.

Intel has worked with the various open source compilers to add support for the return trampoline, and with the OS vendors to use these techniques where appropriate. For Intel® Core™ processors of the Broadwell generation and later, this retpoline mitigation strategy also requires a microcode update to be applied for the mitigation to be fully effective.

3.3 Variant 3: Rogue Data Cache Load Mitigation

For the rogue data cache load method, the operating system software may ensure that privileged pages are not mapped when executing user code in order to protect against user mode access to privileged pages. The OS can establish two paging structure roots (CR3 values) for each user process:

- The "User" paging structure should map all application pages, but only the minimal subset of supervisor pages required for normal processor operation and for transitioning to and from full supervisor mode.
- The "Supervisor" paging structure should map all kernel pages. It may wish to map application pages as well, for convenient access.

This basic dual-page-table approach was previously proposed as a mitigation for side channel attacks on Kernel Address Space Layout Randomization (KASLR) in the "KASLR is Dead: Long Live KASLR" paper and was called KAISER. This approach also mitigates Rogue Data Cache Load. Intel has worked with various OS vendors to enable a dual-page-table approach in their operating systems.

An OS implementing this dual-page-table mitigation may wish to take advantage of the Process Context Identifier (PCID) feature on processors which support it. PCID can greatly reduce the

[2] For details on retpoline, see: https://support.google.com/faqs/answer/7625886

performance cost of TLB flushes caused by frequent reloading of CR3 during user/supervisor mode transitions.

Future Intel processors will also have hardware support for mitigating Rogue Data Cache Load.

3.4 Variant 3a: Rogue System Register Read Mitigation

For malware to compromise security using the rogue system register read method, it must be running locally on a system.

Where applicable, processors may receive microcode updates which ensure that the RDMSR instruction will not speculatively return data when executed at CPL > 0 or when executed by a VMX guest to an MSR for which RDMSR is configured to cause a VM exit.

Future processors may further restrict speculative values returned.

3.5 Variant 4: Speculative Store Bypass Mitigation

Intel recommends using the below mitigations only for managed runtimes or other situations that use language based security to guard against attacks within an address space. The speculative store bypass method mitigation can be accomplished through software modification of affected code or, if that is insufficient, through software setting a new Speculative Store Bypass Disable (SSBD) MSR bit.

As speculative store bypass can only occur when a load is able to execute before an older store with an overlapping address computes its address, an `LFENCE` between that store and the subsequent load is sufficient to prevent this case. Software should be careful to apply this mitigation judiciously to avoid unnecessary performance loss.

Another mitigation is to isolate secrets into a separate address space from code that is relying on language based security (e.g., a managed runtime) to prevent access to those secrets. This process isolation ensures that an attacker who uses the speculative store bypass method (or other speculative execution side channel method) is unable to access those secrets. Protection Keys, described in Section 4.4, can also be useful in providing such isolation.

Other mitigations like inserting register dependencies between a vulnerable load address and the corresponding store address may reduce the likelihood of such an attack being successful.

If the previous methods are not feasible, system software can set the Speculative Store Bypass Disable bit in order to prevent loads from executing before all older store addresses are known. This will also prevent the speculative store bypass method. To minimize performance impact, Intel currently recommends not setting Speculative Store Bypass Disable for OS, VMM or applications that do not rely on language based security.

4 Related Intel Security Features and Technologies

There are security features and technologies, either present in existing Intel products or planned for future products, which reduce the effectiveness of the attacks mentioned in the previous sections.

4.1 Intel® OS Guard

When Intel® OS Guard, also known as Supervisor-Mode Execution Prevention (SMEP), is enabled, the operating system will not be allowed to directly execute application code, even speculatively. This makes branch target injection attacks on the OS substantially more difficult by forcing the attacker to find gadgets within the OS code. It is also more difficult for an application to train OS code to jump to an OS gadget. All major operating systems enable SMEP support by default.

4.2 Execute Disable Bit

The Execute Disable Bit is a hardware-based security feature that can help reduce system exposure to viruses and malicious code. Execute Disable Bit allows the processor to classify areas in memory where application code can or cannot execute, even speculatively. This reduces the gadget space, increasing the difficulty of branch target injection attacks. All major operating systems enable Execute Disable Bit support by default. Applications are encouraged to only mark code pages as executable.

4.3 Control flow Enforcement Technology (CET)

On future Intel processors, Control flow Enforcement Technology will allow limiting near indirect jump and call instructions to only target ENDBRANCH instructions. This feature can reduce the speculation allowed to non-ENDBRANCH instructions. This greatly reduces the gadget space, increasing the difficulty of branch target injection attacks.

Retpoline can interfere with proper usage of CET. Intel recommends that software use CET and enhanced IBRS instead of retpoline where CET is supported.

For additional information on CET, see the Control-flow Enforcement Technology Preview located here: https://software.intel.com/sites/default/files/managed/4d/2a/control-flow-enforcement-technology-preview.pdf.

4.4 Protection Keys

On future Intel processors that have both hardware support for mitigating Rogue Data Cache Load (IA32_ARCH_CAPABILITIES[RDCL_NO]) and protection keys support (CPUID.7.0.ECX[3]), protection keys can limit the data accessible to a piece of software. This can be used to limit the memory addresses that could be revealed by a branch target injection or bound check bypass attack.

4.5 Supervisor-Mode Access Prevention (SMAP)

Supervisor-Mode Access Prevention (SMAP) can be used to limit which memory addresses can be used for a cache based side channel, forcing an application attacking the kernel to use kernel memory space for the side channel. This makes it more difficult for an application to perform the attack on the kernel as it is more challenging for an application to determine whether a kernel line is cached than an application line.

5 Conclusions

Along with other companies whose platforms are potentially impacted by these new methods, Intel has worked with software vendors, equipment manufacturers, and other ecosystem partners to develop software and firmware updates that can protect systems from these methods. End users and systems administrators should check with their software vendors and system manufacturers and apply any available updates as soon as practical.

For malware to compromise security using these methods, it must be running locally on a system. Intel strongly recommends following good security practices that protect against malware in general. Doing so will also help protect against possible exploitation of these analysis methods.

The threat environment continues to evolve. Intel is committed to investing in the security and reliability of our products, and to working constructively with security researchers and others in the industry to help safeguard users' sensitive information. Please see the Intel Security Center[3] for more details. Intel is continuing to investigate architecture and/or microarchitecture changes to combat these types of attacks while maintaining high processor performance.

[3] https://security-center.intel.com/