Intel Analysis of Speculative Execution Side Channels

White Paper

Revision 1.0

January 2018

Document Number: 336983-001

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at www.intel.com.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

This document contains information on products, services and/or processes in development. 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.

Intel disclaims all implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

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.

Intel, and the Intel logo are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others

Contents

§

Revision History

Document Number	Revision Number	Description	Date
336983-001	1.0	Initial release.	January 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/, 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 modern operating systems and susceptibility 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 the methods that have been documented by Google Project Zero and describes the mitigations that Intel is pursuing for each.

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. A disadvantage of speculative execution is that 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 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 measuring microarchitectural properties about the system. Unlike buffer overflows and other security exploits, 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.

Three methods have been identified by Google Project Zero where a cache timing side channel can potentially be used to leak secret information.

2.2.1 Bounds Check Bypass

The *bounds check bypass* method takes advantage of speculative execution after conditional branch instructions. An attacker discovers or causes the creation of 'confused deputy' code which allows the attacker to cause speculative operations to reveal 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 done 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 attacker may be able to exploit that deputy in order to deduce the contents of memory accessible to that deputy but not to the attacker.

2.2.2 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 that code has on the caches to infer data values.

For conditional direct branches, there are only two options on 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 thus the exact methods used may 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, they can use this information to create interference due to aliasing. Similarly, on processors that support Hyperthreading, 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 Rogue Data Cache Load

The final method is the *rogue data cache load*. This 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.

3 Mitigations

Intel has been working closely with the ecosystem, including other processor vendors and software developers, to identify mitigations for the three side channel methods previously described. The mitigation strategy is focused on identifying techniques that can be applicable for both 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 and may decrease the performance cost of other mitigations. See the Related Intel Security Features section for additional details on these security features.

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

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

3.1 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 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 their 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. This 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 introduces the concept of a "return trampoline", also known as "retpoline". 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 ensure support for the return trampoline, and with the OS vendors to ensure support for these techniques. 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 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[1]" 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 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.

[1] KASLR is Dead: Long Live KASLR

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.

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.

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 and protection keys support, 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[2] for more details. Intel is continuing to investigate architecture and/or microarchitecture changes to combat these types of attacks while maintaining high processor performance.

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