Introduction

This document should be read together with the following articles:

- Security Team Spectre (Variant 1 and 2) and Meltdown (Variant 3) Knowledge Base Article
- Security Team Lazy Floating Point Knowledge Base Article (LazyFP)
- Security Team Bounds Check Bypass Store Article (Spectre 1.1 / BCBS)
- Security Team Variant 4 Knowledge Base Article (Variant 4)
- Security Team L1TF Knowledge Base Article (L1TF)
- Security Team MDS Knowledge Base Article (MDS)
- Security Team Mitigation Controls Knowledge Base Article

And the references used here are, among the documents above, the following docs:

- Intel Speculative Execution Side Channel Mitigations (Rev 3.0)
- Intel Analysis of Speculative Execution Side Channels (Rev 4.0)
- Intel Deep Dive Mitigation Overview Side Channel Exploits Linux
- Intel Deep Dive Analysis for L1 Terminal Fault
- Intel Software Guidance for L1 Terminal Fault
- Intel Deep Dive Analysis Microarchitectural Data Sampling
- Using Intel Compilers to Mitigate Speculative Execution Side Channel Issues
- Wikipedia: Kernel Page-Table Isolation
- Wikipedia: Lazy FP state restore
- QEMU Spectre and Meltdown Update
- QEMU Spectre and Meltdown New Update
- Berrange CPU Model Configuration for QEMU/KVM on X86 hosts
- Kernel Admin Guide on HW vulnerabilities - MDS
- Kernel Admin Guide on HW vulnerabilities - MDS

How to read this document ?

If it is written in black, you should read.
If it is in gray, it means it is extra information.
If it is in red, it is a CPU vulnerability.
If it is in green, it is a mitigation technique.
If it is in magenta, it is a CPU flag to advertise Kernel and/or Hypervisor.

Notes:

1) Sometimes the mitigation name is the acronym for the CPU flag, example: Indirect Branch Prediction Barrier (**ibpb**). And sometimes some CPU flag might mitigate more than 1 vulnerability.

2) Downloading the HTML from the wiki (through the proper Download link at the top) and opening it locally will make it better for reading (no CSS themes). You can also print it to PDF if you'd like.

Also, do check articles dates and look for newer ones not yet updated in this document. Newer side channel attack techniques might have been discovered by the time this page was created. If you feel this page should be updated, please send us an e-mail at: ubuntu-server@lists.ubuntu.com mentioning what should be updated and why.

Related CPU Vulnerabilities (CVEs)

1. Side Channel Attacks - Spectre and Meltdown

a) CVE-2017-5753 - Bounds Check Bypass (Variant 1 / Spectre)
b) CVE-2017-5715 - Branch Target Injection (Variant 2 / Spectre)
c) CVE-2017-5754 - Rogue Data Cache Load (Variant 3 / Meltdown)

2. Side Channel Attacks - Others

a) CVE-2018-3665 - Lazy FP Save/Restore (LazyFP)
b) CVE-2018-3693 - Bounds Check Bypass Store (Variant (or Spectre) 1.1 and 1.2 / BCBS)
c) CVE-2018-3640 - Rogue System Register Read (RSRE / Variant 3a)
d) CVE-2018-3639 - Speculative Store Bypass (SSB / Variant 4 / Spectre-NG)

3. L1 Terminal Fault (L1TF)

a) CVE-2018-3615 - Intel SGX (Software Guard Extensions) (Foreshadow / L1TF)

b) CVE-2018-3620 - Operating Systems and System Management Mode (Fault-OS / SMM) (L1TF)
c) CVE-2018-3646 - Virtualization Extensions (L1TF)

4. Microarchitectural Data Sampling (MDS)

a) CVE-2018-12126 - Microarchitectural Store Buffer Data Sampling (MSBDS / Fallout)
b) CVE-2018-12127 - Microarchitectural Load Port Data Sampling (MLPDS / RIDL)
c) CVE-2018-12130 - Microarchitectural Fill Buffer Data Sampling (MFBDS / ZombieLoad)
d) CVE-2019-11091 - Microarchitectural Data Sampling Uncacheable Memory (MDSUM)

This same "per CVE" structure is used in all subsequent sections of this document.
The idea is to allow reader to map:

```
CPU Vulnerability    <->    Firmware/Kernel/OS Mitigation    <->    CPU Flags to Guest
```

CPU Vulnerabilities, HW / Firmware, Kernel and OS Mitigations

1. Side Channel Attacks - Spectre and Meltdown

a) CVE-2017-5753 - Bounds Check Bypass (Variant 1 / Spectre)

Spectre Variant 1 targets 'conditional branch instructions'. The operating system uses conditional branches for processing and validating virtually all data, including untrusted data processed by the kernel. Even code appearing “correct” can potentially be exploited to gain unauthorized access to information.

a.1) __user pointer sanitization

To mitigate this exploit technique in the Linux kernel, we first identify instruction sequences that can be tricked into exploitable behavior. Since a sequence must fit a specific pattern and operate on untrusted data before any potential exploit, not all conditional branches are exploitable. Techniques such as static analysis or manual inspection can identify these sequences.

b) CVE-2017-5715 - Branch Target Injection (Variant 2 / Spectre)

The branch target injection exploit targets a processor’s 'indirect branch predictor'. Indirect branches are used very differently than the conditional branches that the first exploit may target. In Linux, indirect branches are used relatively rarely compared to conditional branches, but they are used in critical locations. In addition, the compiler may insert indirect branches without the programmer ever being aware.

Since the compiler generates these branches, mitigation against this exploit is the most straightforward when the compiler can simply avoid generating vulnerable branch sequences. A software construct called retpoline can be deployed to help ensure that a given indirect branch is resistant to exploitation. The compiler, automatically, or the programmer, manually, can insert the retpoline logic in binary code. Retpoline deliberately steer the processor’s branch prediction logic to a trusted location, preventing a potential exploit from steering them elsewhere.

b.1) OS and Kernel Mitigation: Retpoline

Retpoline needs code (kernel and/or userland) to be recompiled by a compiler with this feature enabled. If not using retpoline, firmware microcode might ALSO mitigate against Spectre Variant 2 on several different platforms.

b.2) Firmware Mitigations:

for Spectre Variant 2 are also provided by different CPU microcodes. Taking amd64 by example, there are MORE THAN 1 mitigation included in more recent microcodes AND about to (or being, or recently done) be included in newer CPUs internal logic.

b.2.1) Enhanced Indirect Branch Restricted Speculation (IBRS): Restricts speculation of indirect branches.

With enhanced IBRS, the predicted targets of indirect branches executed cannot be controlled by software that was executed in a less privileged predictor mode or on another logical processor. As a result, software operating on a processor with enhanced IBRS need not use WRMSR to set IA32_SPEC_CTRL.IBRS after every transition to a more privileged predictor mode. Software can isolate predictor modes effectively simply by setting the bit once.

b.2.2) Single Thread Indirect Branch Predictors (STIBP): Prevents indirect branch predictions from being controlled by the sibling HW thread.

STIBP is an indirect branch control mechanism that restricts the sharing of branch prediction between logical processors on a core. ... as the logical processors sharing a core may share indirect branch predictors, allowing one logical processor to control the predicted targets of indirect branches by another logical processor of the same core

Enabling IBRS prevents software operating on one logical processor from controlling the predicted targets of indirect branches executed on another logical processor. For that reason, it is not necessary to enable STIBP when IBRS is enabled.

b.2.3) Indirect Branch Predictor Barrier (IBPB): Ensures earlier code's behavior does not control later indirect branch predictions. Complements Enhanced IBRS.

IBPB is an indirect branch control mechanism that establishes a barrier, preventing software that executed before the barrier from controlling the predicted targets of indirect branches executed after the barrier on the same logical processor. A processor supports IBPB if it enumerates CPUID.(EAX=7H,ECX=0):EDX[26] as 1.

IBPB does not define a new mode of processor operation that controls the branch predictors, unlike Indirect Branch Restricted Speculation (IBRS) and Single Thread Indirect Branch Predictors (STIBP). As a result, it is not enabled by setting a bit in the IA32_SPEC_CTRL MSR. Instead, IBPB is a “command” that software executes when necessary.

c) CVE-2017-5754 - Rogue Data Cache Load (Variant 3 / Meltdown)

The rogue data cache load exploit targets a processor’s speculative data loading mechanisms. Even though the processor’s access control may protect a piece of data, it may still be read speculatively before an illegal access is determined to exist. The first two exploits require manipulating the kernel to do something for an attacker, while this exploit occurs entirely within code which is under the control of an attacker. This means we cannot modify the kernel to mitigate this exploit, we must fundamentally change where kernel data is available.

c.1) Kernel Mitigation: Kernel Page Table Isolation

The mitigation for this is conceptually very simple: instead of relying on a processor’s access-control mechanisms to protect data, simply remove the data instead.

In Linux, this mitigation is referred to as Kernel Page Table Isolation (PTI). This mitigation removes the data from the reach of exploits by having the kernel maintain two independent copies of the hardware page tables. One copy contains the minimal set of data and code needed to run an application and enter or exit the kernel, but it does not contain valuable kernel data. The other set of page tables, active only while the kernel is running, contains everything needed for the kernel to function normally, including its private data.

2. Side Channel Attacks - Others

a) CVE-2018-3665 - Lazy FP Save/Restore (LazyFP)

Lazy FPU state leak, or Lazy FP State Restore, or LazyFP, affects only Intel Core CPUs. It is a combination of flaws in the speculative execution technology presented in some CPUs, and how certain the OS handle context switching on the floating point unit. Basically a local process can leak the content of the FPU register to another process.

a.1) Kernel Mitigation

Intel recommends OS to use Eager FP state restore in lieu of Lazy FP state restore. Linux Kernel does that since kernel 4.5. Older kernels running on processors that support the xsaveopt instruction are also not affected. You can verify if your system has support for xsaveopt by locating the "xsaveopt" feature listed in the flags section of the /proc/cpuinfo file. No particular mitigation for QEMU/KVM.

b) CVE-2018-3693 - Bounds Check Bypass Store (Variant (or Spectre) 1.1 and 1.2 / BCBS)

Many Intel processors use memory disambiguation predictors that allows loads to be executed speculatively before it is known whether the load’s address overlaps with a preceding store’s address. This may happen if a store’s address is unknown when the load is ready to execute. If the processor predicts that the load address will not overlap with the unknown store address, the load may execute speculatively. However, if there was indeed an overlap, then the load may consume stale data. When this occurs, the processor will re-execute the load to ensure a correct result.

Through the memory disambiguation predictors, an attacker can cause certain instructions to be executed speculatively and then use the effects for side channel analysis.

b.1) Software Based Mitigations

b.1.1) Process Isolation

To run untrusted code in different processes, so the address space is not shared among trusted and untrusted code.

b.1.2) LFENCE (requires new firmware/microcode)

Software can insert a LFENCE between a store and a subsequent load to prevent the load from executing before the previous store's address is known. This should be applied only in places with realistic risk, as it incurs in performance penalty.

b.2) Firmware Mitigation: Speculative Store Bypass Disable (SSBD)

c) CVE-2018-3640 - Rogue System Register Read (RSRE / Variant 3a)

The rogue system register read method, 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.

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.

c.1) Firmware Mitigation: Speculative Store Bypass Disable (SSBD)

d) CVE-2018-3639 - Speculative Store Bypass (SSB / Variant 4 / Spectre-NG)

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.

d.1) *OS and Firmware Mitigation:* Speculative Store Bypass Disable (SSBD)

SSBD prevents a load from executing speculatively until the address of all older stores are known. In Ubuntu, SSBD is OFF by default because it is not needed by most programs and carries a notable performance impact. A prctl() has been added (PR_SPEC_STORE_BYPASS) that enables developers to opt into the mitigation on a per process basis.

Applications using seccomp filters will be implicitly opted into the mitigations (snaps, processes inside LXD containers, sandboxed firefox and chromium processes, will have SSBD mitigation enabled out of the box).

3. L1 Terminal Fault (L1TF)

When a program attempts to access data in memory, the logical memory address is translated to a physical address by the hardware. Accessing a logical or linear address that is not mapped to a physical location on the hardware will result in a terminal fault.

Once the fault is triggered, there is a gap, before resolution, where the processor will use speculative execution to try to load data. During this time, the processor could speculatively access the level 1 data cache (L1D), potentially allowing side-channel methods to infer information that would otherwise be protected.

Because the resulting probed physical address is not a true translation of the virtual address, the resulting address is not constrained by various memory range checks or nested translations. Specifically:

- Intel SGX protected memory checks are not applied.
- Extended Page Table guest physical to host physical address translation is not applied.
- SMM protected memory checks are not applied.

Mitigations bellow server for different L1TF vulnerabilities sub-types.

3.1) Using Newer CPUs or Firmware: Check Intel Documentation

3.2) Kernel Mitigation: Removing Secrets from L1D

Data that might be leaked by an L1TF exploit must be present in the L1D while malicious code executes. When transitioning to less-privileged code, removing data from the L1D mitigates exploits that might be launched in the less-privileged code.

3.3) Page Table Entries Inversion

OS is responsible for mitigating against exploitation of paging structure entries by malicious applications. To do this, the OS can ensure that vulnerable Page Table Entries (PTEs) refer only to specifically-selected physical addresses, such as those addresses outside of available cached memory or addresses that do not contain secrets.

There are four typical cases that need mitigation in an OS:

3.2.1) Pages with no valid mappings.
3.2.2) Pages that have been written to other storage (swapped out)
3.2.3) Pages where the application has requested that the OS disable access.
3.2.4) Pages in transitional states where the OS needs to temporarily block access.

In the first case, OSes may use an all-zero paging structure entry to represent linear addresses with no physical mapping (including the Page Size Extension (PSE) bit7 where supported) while ensuring that the 4 KB page frame starting at physical address 0 contains no secrets.

For the other three cases, the OS should make the PTEs refer to invalid memory addresses, as described in the following sections.

Disabling EPT for virtual machines provides full mitigation for L1TF even with SMT enabled, because the effective page tables for guests are managed and sanitized by the hypervisor. Though disabling EPT has a significant performance impact especially when the Meltdown mitigation KPTI is enabled.

3.4) SMT Disablement

To prevent the SMT issues of L1TF it might be necessary to disable SMT completely. Disabling SMT can have a significant performance impact, but the impact depends on the hosting scenario and the type of workloads. The impact of disabling SMT needs also to be weighted against the impact of other mitigation solutions like confining guests to dedicated cores.

a) CVE-2018-3615 - Intel SGX (Software Guard Extensions) (Foreshadow / L1TF)
(does not affect Ubuntu)

The enclave-to-enclave (E2E) method is a sub-variant of the L1TF method. E2E may expose memory in one Intel SGX enclave to software that is running in a different Intel SGX enclave on the same core.

a.1) No mitigation needed for Ubuntu Linux,

b) CVE-2018-3620 - Operating Systems and System Management Mode (Fault-OS / SMM) (L1TF)

SMM is a special processor mode used by BIOS. SMM software must rendezvous all logical processors both on entry to, and exit from, SMM to ensure that a sibling logical processor does not reload data into the L1D after the automatic flush. We believe most SMM software already does this. This will ensure that non-SMM software does not run while lines that belong to SMM are in the L1D. Such SMM implementations do not require any software changes to be fully mitigated for L1TF.

b.1) Using Newer CPUs or Firmware: *Check* *Intel Documentation*

The SMRR MSRs are used to protect **SMM** and will prevent non-**SMM** code from bringing SMM lines into the L1D. Processors that enumerate L1D_FLUSH and are affected by L1TF will automatically flush the **L1D** during the RSM instruction that exits **SMM**.

b.2) Check item (3.3): Kernel Mitigation: *Removing Secrets from L1D*

An updated kernel removes the possibility for unintended memory exposure between processes within the same operating system environment by adjusting page table entries for not-present pages to point to uncacheable memory. The mitigation for this vulnerability introduces negligible performance impacts.

c) CVE-2018-3646 - Virtualization Extensions (L1TF)

This issue only affects virtualization hypervisors and does not impact systems where virtualization is not used. Anything mentioned here is specific to KVM in the Linux Kernel.

Processors that implement Intel HT share the L1D between all logical processors (hyperthreads) on the same physical core. This means that data loaded into the L1D by one logical processor may be speculatively accessed by code running on another logical processor. Disabling hyperthreading does not in itself provide mitigation for L1TF.

c.1) Check item (3.3): Kernel Mitigation: *Removing Secrets from L1D*

An updated kernel will flush the L1 data cache in some specific scenarios when the host enters the guest. Flushing the L1 data cache is an expensive operation that negates the performance improvements of caching so flushing the L1 data cache is performed selectively to protect certain code paths that could cause unintended memory exposure to a malicious guest.

Optimized L1 data cache flushing is available via intel-microcode updates (L1D_FLUSHcapability). The updated kernels implement a software fallback cache flushing mechanism for processors that have not received microcode updates.

When guests are trusted or belong to the same security domain, no mitigation is needed. However, VMMs generally allow untrusted guests to place arbitrary translations in the guest paging structure entries because VMMs assume any entries will be translated with VMM-controlled EPT. But EPT translation is not performed in the case of an L1 terminal fault.

This means a malicious guest OS may be able to set up values in its paging structure entries that attack arbitrary host addresses, theoretically enabling an exploit to access any data present in the L1D on the same physical core as the malicious guest. For this reason, VMM mitigations are focused on ensuring secret data is not present in the L1D when executing guests.

4. Microarchitectural Data Sampling (MDS)

This vulnerability affects a wide range of Intel processors.

It was discovered that memory contents previously stored in microarchitectural buffers of an Intel CPU core may be exposed to a malicious process that is executing on the same CPU core via a speculative execution side-channel. A local attacker could access the stale contents of store buffers, load ports, and fill buffers which may contain data belonging to another process or data that originated from a different security context. As a result, unintended memory exposure can occur between userspace processes, between the kernel and userspace, between virtual machines, or between a virtual machine and the host environment.

MDS differs from other recent speculative execution side-channel attacks in that the attacker cannot target specific data. The attacker can periodically sample the contents in the buffers but does not have control over the data that is present in the buffers when the sample is taken. Therefore, additional work is required to fully collect and reconstruct the data into a meaningful data set.

The mitigation for microarchitectural data sampling issues includes clearing store buffers, fill buffers, and load ports before transitioning to possibly less privileged execution entities.

a) CVE-2018-12126 - Microarchitectural Store Buffer Data Sampling (MSBDS / Fallout)

When performing store operations, processors write data into a temporary microarchitectural structure called the store buffer. This enables the processor to continue to execute instructions following the store operation, before the data is written to cache or main memory. I/O writes (for example, OUT) are also held in the store buffer.

When a load operation reads data from the same memory address as an earlier store operation, the processor may be able to forward data to the load operation directly from the store buffer instead of waiting to load the data from memory or cache. This optimization is called store-to-load forwarding.

Under certain conditions, data from a store operation can be speculatively forwarded from the store buffer to a faulting or assisting load operation for a different memory address. It is possible that a store does not overwrite the entire data field within the store buffer due to either the store being a smaller size than the store buffer width, or not yet having executed the data portion of the store. These cases can lead to data being forwarded that contains data from older stores. Because the load operation will cause a fault/assist and its results will be discarded, the forwarded data does not result in incorrect program execution or architectural state changes. However, malicious actors may be able to forward this speculative-only data to a disclosure gadget in a way that allows them to infer this value.

b) CVE-2018-12127 - Microarchitectural Load Port Data Sampling (MLPDS / RIDL)

Processors use microarchitectural structures called load ports to perform load operations from memory or I/O. During a load operation, the load port receives data from the memory or I/O system, and then provides that data to the register file and younger dependent operations. In some implementations, the writeback data bus within each load port can retain data values from older load operations until younger load operations overwrite that data. Since speculatively executed load operations may receive stale data values from the store buffer or fill buffer, the retained values may also hold store values.

c) CVE-2018-12130 - Microarchitectural Fill Buffer Data Sampling (MFBDS / ZombieLoad)

A fill buffer is an internal structure used to gather data on a first level data cache miss. When a memory request misses the L1 data cache, the processor allocates a fill buffer to manage the request for the data cache line. The fill buffer also temporarily manages data that is returned or sent in response to a memory or I/O operation. Fill buffers can forward data to load operations and also write data to the data cache. Once the data from the fill buffer is written to the cache (or otherwise consumed when the data will not be cached), the processor deallocates the fill buffer, allowing that entry to be reused for future memory operations.

Fill buffers may retain stale data from prior memory requests until a new memory request overwrites the fill buffer. Under certain conditions, the fill buffer may speculatively forward data, including stale data, to a load operation that will cause a fault/assist. This does not result in incorrect program execution because faulting/assisting loads never retire and therefore do not modify the architectural state. However, a disclosure gadget may be able to infer the data in the forwarded fill buffer entry through a side channel timing analysis.

d) CVE-2019-11091 - Microarchitectural Data Sampling Uncacheable Memory (MDSUM)

Data accesses that use the uncacheable (UC) memory type do not fill new lines into the processor caches. On processors affected by Microarchitectural Data Sampling Uncachable Memory (MDSUM), load operations that fault or assist to uncacheable memory may still speculatively see the data value from those core or data accesses. Because uncacheable memory accesses still move data through store buffers, fill buffers, and load ports, and those data values may be speculatively returned on faulting or assisting loads, malicious actors can observe these data values through the MSBDS, MFBDS, and MLPDS mechanisms discussed above.

- Mitigations for (a), (b), (c) and (d):

  The vulnerability is **NOT** present on:

  - Processors from AMD, Centaur and other non Intel vendors
  - Older processor models, where the CPU family is < 6
  - Some Atoms (Bonnell, Saltwell, Goldmont, GoldmontPlus)
  - Intel processors w/ ARCH_CAP_MDS_NO bit set in the IA32_ARCH_CAPABILITIES MSR.

CASE 01) Non SMT (Simultaneous Multithreading) CPUs:

There are two methods to overwrite the microarchitectural buffers affected by MDS: MD_CLEAR functionality and software sequences.

4.1) Using Newer CPUs or Firmware

Through IA32_ARCH_CAPABILITIES MSR it is possible to see if the current CPU has the RDCL_NO (Not susceptible to Rogue Data Cache Load) feature. If it has, then the processor is also not affected by MFBDS.

Recently IA32_ARCH_CAPABILITIES MSR was expanded to sinalize the MDS_NO feature. A value of 1 indicates that processor is not affected by MFBDS/MSBDS/MLPDS/MDSUM.

Note that MFBDS is mitigated if either the RDCL_NO or MDS_NO bit (or both) are set. Some existing processors may also enumerate either RDCL_NO or MDS_NOonly after a microcode update is loaded.

Intel will release microcode updates and new processors that enumerate MD_CLEAR functionality. On processors that enumerate MD_CLEAR, the VERW instruction or L1D_FLUSH command should be used to cause the processor to overwrite buffer values that are affected by MDS, as these instructions are preferred to the software sequences.

4.1.1) MD_CLEAR + QEMU: L1D_FLUSH support when entering vCPU

The VMM can execute either the VERW instruction or the L1D_FLUSH command before entering a guest VM. This will overwrite protected data in the buffers that could belong to the VMM or other VMs. VMMs that already use the L1D_FLUSH command before entering guest VMs to mitigate L1TF may not need further changes beyond loading a microcode update that enumerates MD_CLEAR.

4.1.2) MD_CLEAR + Kernel Mitigations:
VERW instr on ring 0 <-> 3 and c-states transitions

The OS can execute the VERW instruction to overwrite any protected data in affected buffers when transitioning from ring 0 to ring 3. This will overwrite protected data in the buffers that could belong to the kernel or other applications.

4.1.3) MD_CLEAR + System Management Mode (SMM)

Exposure of system management mode (SMM) data to software that subsequently runs on the same logical processor can be mitigated by overwriting buffers when exiting SMM. On processors that enumerate MD_CLEAR, the processor will automatically overwrite the affected buffers when the RSM instruction is executed.

4.2) NO MD_CLEAR + Software Based Mitigation

On processors that do not enumerate the MD_CLEAR functionality, certain instruction sequences may be used to overwrite buffers affected by MDS. These sequences are described in detail in the Software sequences to overwrite buffers section.

CASE 02) For SMT (Simultaneous Multithreading) capable CPUs

4.3) Sibling threads and OS Mitigations

Even with MD_CLEAR capable firmwares and new CPUs, the OS must employ two different methods to prevent a thread from using MDS to infer data values used by the sibling thread. The first (**group scheduling**) protects against user vs. user attacks. The second (**synchronized entry**) protects kernel data from attack when one thread executes kernel code by an attacker running in user mode on the other thread.

4.3.1) HW Disable

A HW only method method to prevent the sibling thread from inferring data values through MDS is to disable SMT either through the BIOS or by having the OS only schedule work on one of the threads.

4.3.2) Group Scheduling

The OS can prevent a sibling thread from running malicious code when the current thread crosses security domains. The OS scheduler can reduce the need to control sibling threads by ensuring that software workloads sharing the same physical core mutually trust each other (for example, if they are in the same application defined security domain) or ensuring the other thread is idle.

4.3.3) Synchronized Entry

The OS needs to take action when the current hardware thread makes transitions from user code (application code) to the kernel code (ring 0 mode). This can happen as part of syscall or asynchronous events such as interrupts, and thus the sibling thread may not be allowed to execute in user mode because kernel code may not trust user code.

4.3.4) Virtual Machine Manager (VMM)

Processors that enumerate MD_CLEAR have enhanced the L1D_FLUSH command to also overwrite the microarchitectural structures affected by MDS. This can allow VMMs that have mitigated L1TF through group scheduling and through using the L1D_FLUSH command to also mitigate MDS.

The VMM mitigation may need to be applied to processors that are not affected by L1TF (RDCL_NO is set) but are affected by MDS (MDS_NO is clear). VMMs on such processors can use VERW instead of the L1D_FLUSH command.

VMMs that have implemented the **L1D flush** using a software sequence should use a VERW instruction to overwrite microarchitectural structures affected by MDS.

QEMU/KVM ONLY - Mitigations and CPU Flags

In order to protect the guest kernel from a malicious userspace, updates are also needed to the guest kernel and, depending on the processor architecture, to QEMU.

QEMU configures the hypervisor to emulate a specific processor model. For x86, QEMU has to be aware of new CPUID bits introduced by the microcode update (next section clarifies CPU flags), and it must provide them to guests depending on how the guest is configured.

Once updates are provided, live migration to an updated version of QEMU will not be enough to protect guest kernel from guest userspace. Because the virtual CPU has to be changed to one with the new CPUID bits, the guest will have to be restarted.

CPU features are advertised, by the CPU to the OS Kernel, through the CPUID instruction. Depending on how the CPU registers are set, different type of information regarding CPU supported features are returned by the CPU itself. To extend the capability of receiving more data, than the ones available through the CPU register on the return of the instruction, there is also the possibility that some of the return is given through MSRs (Model-Specific Registers): memory mapped internal to CPU registers.

With a specific CPUID instruction, available CPU mitigation supportability is reported through a CPU register:

Bit 26: **IBRS** and **IBPB** are supported (Spectre Variants)
Bit 27: **STIBP** is supported (Spectre Variants)
Bit 28: **L1D_FLUSH** is supported (L1TF)
Bit 29: **IA32_ARCH_CAPABILITIES** supported (will provide the list bellow in a MSR)
Bit 30: -
Bit 31: **SSBD** supported (SSB / Variant 4)

The MSR used for the IA32_ARCH_CAPABILITIES feature, listing other CPU features available for the CPU being queried, can list the following CPU features/mitigations:

Bit 00: **RDCL_NO** (CPU not susceptible to RDCL - Rogue Data Cache Load) - Meltdown / Variant 3
Bit 01: **IBRS_ALL** (Processor supports Enhanced IBRS) - Better/Faster IBRS (less overhead)
Bit 02: **RSBA** (Processor supports RSB Alternate) - Spectre Variant 2 / RSB underflow
Bit 03: **SKIP_L1DFL_VMENTRY** (1 means CPU need not to Flush L1D on vCPU VM Entry)
Bit 04: **SSB_NO** (Processor is NOT susceptible to SSB Speculative Store Bypass)

NOTE: Some of the CPU features have the same name as the Mitigation Technique being used for the security vulnerability.

1. Side Channel Attacks - Spectre and Meltdown

QEMU contains the mitigation functionality for x86 guests, along with additional mitigation functionality for pseries and s390x guests. Please note that QEMU/KVM has at least the same requirements as other unprivileged processes running on the host with regard to Spectre/Meltdown mitigation.

What is being addressed here is enabling a guest operating system to enable the same (or similar) mitigations to protect itself from unprivileged guest processes running under the guest operating system.

- Vulnerabilities (CVEs) & Mitigation Notes

a) CVE-2017-5753 - Bounds Check Bypass (Variant 1 / Spectre)

a.1) __user pointer sanitization on both, VMM/Hypervisor Kernel & Guest Kernel.

b) CVE-2017-5715 - Branch Target Injection (Variant 2 / Spectre)

Among the three vulnerabilities, CVE-2017-5715 is notable because it allows guests to read potentially sensitive data from hypervisor memory. Patching the host kernel is sufficient to block attacks from guests to the host but not from guest user to guest kernel (guest kernel also needs mitigation).

b.1) Host Kernel and KVM: Retpoline

Ubuntu kernel is compiled with retpoline support.

b.2) Firmware Mitigations:

b.2.1) Enhanced Indirect Branch Restricted Speculation (IBRS): Restricts speculation of indirect branches.

Ubuntu kernel is compiled with IBRS support. Virtual CPU flag **spec-ctrl** should be used to advertise **ibrs** capability to Virtual Machine Guests.

b.2.2) Single Thread Indirect Branch Predictors (STIBP): Prevents indirect branch predictions from being controlled by the sibling HW thread.

Virtual CPU flag **stipb** COULD be used to advertise the **stipb** capability to Virtual Machine Guests, but **spec-ctrl** & **ibpb** CPU capabilities are enough - and have better performance - to also mitigate STIBP.

b.2.3) Indirect Branch Predictor Barrier (IBPB): Ensures earlier code's behavior does not control later indirect branch predictions. Complements Enhanced IBRS.

Ubuntu kernel is compiled with IBPB support. Virtual CPU flag **ibpb** should be used to advertise **ibpb** capabilities to Virtual Machine Guests.

c) CVE-2017-5754 - Rogue Data Cache Load (Variant 3 / Meltdown)

c.1) Kernel Mitigation: Kernel Page Table Isolation (KPTI)

Meltdown flaw does not allow a malicious guest to read the contents of hypervisor memory. Fixing it only requires that the operating system separates the user and kernel address spaces (known as **page table isolation**), which can be done separately on the host and the guests (already covered in previous section).

Because the cost of Kernel Page Table Isolation is high, it is recommended to advertise the **pcid** CPU feature to guests, so they can mitigate the cost of the Meltdown fix with this feature.

- CPU flags to advertise CPUs (or Microcodes) have mitigations

Intel x86 hosts

There are 3 additional **CPU flags** associated with Spectre/Meltdown mitigation:

```
spec-ctrl        Indirect Branch Restricted Speculation (IBRS)
ibpb             Indirect Branch Prediction Barrier (IBPB)
pcid             Process Context Identifiers (PCID)
```

AMD x86 hosts

```
ibpb             Indirect Branch Prediction Barrier (IBPB)
```

These flags expose additional functionality made available through new microcode updates for certain Intel/AMD processors that can be used to mitigate various attack vectors related to Spectre. Having those flags mean you're mitigated through CPU and/or CPU microcodes.

pseries

There are 3 tri-state-machine options/capabilities relating to Spectre/Meltdown mitigation:

```
cap-cfpc         Cache Flush on Privilege Change
cap-sbbc         Speculation Barrier Bounds Checking
cap-ibs          Indirect Branch Serialisation
```

Again, having those mean you're mitigated through CPU and/or CPU microcodes.

s390x

There are 2 CPU feature bits relating to Spectre/Meltdown:

```
bpb             Branch Prediction Blocking
ppa15           PPA15 is installed
```

A CPU having those flags possibly mean you're mitigated through CPU and/or CPU microcodes. Instead of trying to map one or another, check last session of this document for the vulnerability scanner.

Check Spectre and Meltdown Mitigation Controls to understand how to opt in or out from existing mitigations, or how to let the Host (or Guest) kernel to decide which mitigation to use. That's why its important to understand existing CPU flags and why they should be passed correctly to VMs: to let Guest kernels to decide the best, and likely most performant, mitigation available for a vulnerability.

2. Side Channel Attacks - Others

- Vulnerabilities (CVEs) & Mitigation Notes

a) CVE-2018-3665 - Lazy FP Save/Restore (LazyFP)

a.1) Kernel Mitigation

No particular mitigation needed for QEMU/KVM. Guests kernels would ALSO have to make sure to use Eager FP Save Restore instead of Lazy, OR check if xsaveopt CPU feature is available for the vCPUs, which would be already enough not to suffer from this vulnerability.

b) CVE-2018-3693 - Bounds Check Bypass Store (Variant (or Spectre) 1.1 and 1.2 / BCBS)

b.1) Software Based Mitigations

b.1.1) Process Isolation

N/A

b.1.2) LFENCE (requires new firmware/microcode)

N/A

b.2) Firmware Mitigation: Speculative Store Bypass Disable (SSBD)

N/A

c) CVE-2018-3640 - Rogue System Register Read (RSRE / Variant 3a)

c.1) Firmware Mitigation.

N/A

d) CVE-2018-3639 - Speculative Store Bypass (SSB / Variant 4 / Spectre-NG)

d.1) *Firmware Mitigation:* Speculative Store Bypass Disable (SSBD)

SSBD prevents a load from executing speculatively until the address of all older stores are known. If CPU/Microcode supports it, the CPU flag should be advertised to virtual machines.

The CPU feature ssbd should be advertised to virtual CPUs so Virtual Machine Guests know that disabling Speculative Store Bypass (SSB) is possible. Like explained in the previous session of this document, SSBD can be enabled/disabled through seccomp or prctl.

- CPU flags to advertise CPUs (or Microcodes) have mitigations

Intel x86 hosts

There are 1 additional CPU flags associated with Spectre/Meltdown mitigation:

```
ssbd            Speculative Store Bypass Disable (SSBD)
```

Requires the host CPU microcode to support this feature before it can be used for guest CPUs.

AMD x86 hosts

There are 2 additional CPU flags associated with Spectre/Meltdown mitigation:

```
virt-ssbd       Speculative Store Bypass Disable (lower performance)
amd-ssbd        Speculative Store Bypass Disable (best performance)
amd-no-ssb      Host is not vulnerable to Variant 4 (future CPUs)
```

virt-ssbd provides a lower performance than amd-ssbd but both should be provided to virtual machines for compatibility reasons, as some of virtual machines might not support amd-ssbd and will have to stick with virt-ssbd mitigation.

Check Spectre and Meltdown Mitigation Controls to understand how to choose the best way to enable the SSBD (disable by default, opt-in via prctl() ? seccomp implicitly opt-in with prctl() enabled threads ? All enabled ? SSBD disabled ?). By informing the VM Guests about the ssbd CPU feature, they might opt in or out this mitigation also.

NOTE: When ENABLING SSBD you're actually DISABLING the CPU SSB feature (Speculative Store Bypass), which is where the vulnerability relies. When passing the ssbd CPU feature, you're just advertising the capability of disabling the CPU SSB feature to the guest.

3. L1 Terminal Fault (L1TF)

a) CVE-2018-3615 - Intel SGX (Software Guard Extensions) (Foreshadow / L1TF)
b) CVE-2018-3620 - Operating Systems and System Management Mode (Fault-OS / SMM) (L1TF)
c) CVE-2018-3646 - Virtualization Extensions (L1TF)

- Vulnerabilities (CVEs) & Mitigation Notes

Mitigations bellow server for different L1TF vulnerabilities sub-types.

3.1) Using Newer CPUs or Firmware: Check Intel Documentation

Processors that have the RDCL_NO bit set to one (1) in the IA32_ARCH_CAPABILITIES MSR are NOT susceptible to the L1TF speculative execution side channel. On such processors, none of the following mitigations are required. All virtual machines should be warned about the existence of the rdctl-no CPU feature so their kernels can opt-out from deeper L1D flushes.

3.2) Kernel Mitigation: Removing Secrets from L1D

L1D Flush is supported by Ubuntu Kernel and can be "accelerated" by newer CPUs and or Firmwares containing the L1D_FLUSH CPU feature.

> The **IA32_FLUSH_CMD MSR** is more precise and does not flush higher cache levels, thereby limiting its impact only to the L1 cache on the physical core executing the flush itself.

For Hypervisor Hosts, the SKIP_L1DFL_VMENTRY CPU feature, from IA32_ARCH_CAPABILITIES MSR, explained in the beginning of this session, also tells the Kernel that there is no need to flush the **L1D** cache when entering into a vCPU context (**VMENTRY**), as the CPU is mitigated for L1TF.

3.3) Page Table Entries Inversion

N/A (already done by the hypervisor kernel, guest kernel also has to have this feature to avoid guest user <-> guest kernel attacks).

Disabling EPT for virtual machines provides full mitigation for L1TF even with SMT enabled, because the effective page tables for guests are managed and sanitized by the hypervisor. **Though disabling EPT has a significant performance impact especially when the Meltdown mitigation KPTI is enabled.**

3.4) SMT Disablement

If SMT is not supported by the processor or disabled in the BIOS or by the kernel, it's only required to enforce L1D flushing on VM_ENTER.

- CPU flags to advertise CPUs (or Microcodes) have mitigations

Intel x86 hosts

There are 2 additional CPU flags associated with L1TF vulnerabilities:

```
rdctl-no              Processor is not susceptible to Rogue Data Cache Load (RDCL)
skip-l1dfl-vmentry     Skip L1D_FLUSH whenever there is a VM_ENTRY
```

Both CPU features come from a another CPU feature available in newer CPUs called IA32_ARCH_CAPABILITIES, which is explained in the beginning of this topic (**"QEMU/KVM Mitigations and CPU Flags"**), responsible to enumerate HW mitigations available in newer CPUs. If they're present in your CPU, you should definitely pass them to the Virtual Machine Guests, but always thinking about live migrations among Hosts with different CPU types (and available CPU features).

Always flushing the cache will negatively impact performance in a manner that's dependent on the workload inside of the virtual machine while never flushing the cache will make your system vulnerable to CVE-2018-3646. Use the "kvm-intel.vmentry_l1d_flush=always" or "kvm-intel.vmentry_l1d_flush=never" kernel command line options to change the default persistently across reboots. Write "always", "cond" (the default), or "never" to /sys/module/kvm_intel/parameters/vmentry_l1d_flush to temporarily change the behavior.

Full protection (no minimal realistic leak from L1D cache) from the Hyper-Thread based attack can be achieved in one of two ways:

3.1) The first option requires that Hyper-Threads be disabled in the system's BIOS, by booting with the **"nosmt"** kernel command line option, or by writing **"off"** or **"forceoff"** to **/sys/devices/system/cpu/smt/control** (this file is not persistent across reboots).

3.2) The second option involves restricting guests to specific CPU cores that are not shared with the host or other guests considered to be in different trust domains. This option is more difficult to configure and may still allow a malicious guest to gain some information from the host environment.

Check the Linux Kernel Admin Guide - L1TF Mitigation Control Cmd Line to check if you should do a full L1TF mitigation (disabling SMT and enabling unconditional L1D flushing, or disabling SMT and no L1D flushing, or disabling SMT only). Check the same document for the KVM module parameters (enabling or not vmentry_l1d_flush, like explained previously).

Final choice on which mitigations you are going to use will depend how much you trust the code running in your virtual machines.

4. Microarchitectural Data Sampling (MDS)

a) CVE-2018-12126 - Microarchitectural Store Buffer Data Sampling (MSBDS / Fallout)
b) CVE-2018-12127 - Microarchitectural Load Port Data Sampling (MLPDS / RIDL)
c) CVE-2018-12130 - Microarchitectural Fill Buffer Data Sampling (MFBDS / ZombieLoad)
d) CVE-2019-11091 - Microarchitectural Data Sampling Uncacheable Memory (MDSUM)

- Mitigations for (a), (b), (c) and (d):

CASE 01) Non SMT (Simultaneous Multithreading) CPUs:

There are two methods to overwrite the microarchitectural buffers affected by MDS: MD_CLEAR functionality and software sequences.

4.1) Using Newer CPUs or Firmware

Through IA32_ARCH_CAPABILITIES MSR it is possible to see if the current CPU has the RDCL_NO (Not susceptible to Rogue Data Cache Load) feature. If it has, then the processor is also not affected by MFBDS. With that in mind, passing rdctl-no CPU feature to the virtual machine guest vCPUs is appropriate.

Recently IA32_ARCH_CAPABILITIES MSR was expanded to sinalize the MDS_NO feature. A value of 1 indicates that processor is not affected by MFBDS/MSBDS/MLPDS/MDSUM. For this recent CPU flag, the vCPU used feature is mds-no.

Note that MFBDS is mitigated if either the RDCL_NO or MDS_NO bit (or both) are set, but not MSBDS/MLPDS/MDSUM, those last need MDS_NO feature. Some existing processors may also enumerate either RDCL_NO or MDS_NO only after a microcode update is loaded.

Intel will release microcode updates and new processors that enumerate MD_CLEAR functionality. On processors that enumerate MD_CLEAR, the VERW instruction or L1D_FLUSH command should be used to cause the processor to overwrite buffer values that are affected by MDS, as these instructions are preferred to the software sequences, to mitigate the vulnerabilities.

4.1.1) MD_CLEAR + QEMU: L1D_FLUSH support when entering vCPU

The VMM can execute either the VERW instruction or the L1D_FLUSH command before entering a guest VM. This will overwrite protected data in the buffers that could belong to the hypervisor or other VMs. Hypervisors that already use the L1D_FLUSH command before entering guest VMs to mitigate L1TF may not need further changes beyond loading a microcode update that enumerates MD_CLEAR.

Some new CPUs might also sinalize the SKIP_L1DFL_VMENTRY, which will indicate that there is no need for the hypervisor to flush the L1D cache on a VM_ENTRY.

4.1.2) MD_CLEAR + Kernel Mitigations

N/A

4.1.3) MD_CLEAR + System Management Mode (SMM)

N/A

4.2) NO MD_CLEAR + Software Based Mitigation

CASE 02) For SMT (Simultaneous Multithreading) capable CPUs

4.3) Sibling threads and OS Mitigations

4.3.1) HW Disable

THE ONLY 100% effective method to prevent the sibling thread from inferring data values through MDS is to disable SMT, either through the BIOS, or by having the hypervisor kernel to only schedule vCPUs in one of the HW threads. All other methods can't guarantee 100% the hypervisor guests won't be able to "sniff" L1D cache from the hypervisor and/or another guests.

4.3.2) Group Scheduling

N/A

4.3.3) Synchronized Entry

N/A

4.3.4) Virtual Machine Manager (VMM)

Processors that enumerate MD_CLEAR have enhanced the L1D_FLUSH command to also overwrite the microarchitectural structures affected by MDS. This can allow VMMs that have mitigated L1TF through **group scheduling** and through using the L1D_FLUSH command to also mitigate MDS.

The hypervisor mitigation may need to be applied to processors that are not affected by L1TF (RDCL_NO is set) but are affected by MDS (MDS_NO is clear). Hypervisor on such processors can use VERW instead of the L1D_FLUSH command.

VMMs that have implemented the L1D flush using a software sequence should use a VERW instruction to overwrite microarchitectural structures affected by MDS.

- CPU flags to advertise CPUs (or Microcodes) have mitigations

Intel x86 hosts

All the features bellow can be passed to the virtual machine guest vCPUs in order to inform about host's HW features for mitigations if not using "cpu mode='host-passthrough'" (specially when dealing with minimum CPU features available on hosts and live migration issues due to that).

mds-no	Processor is not susceptible Microarch. Data Sampling (MDS)
md-clear	L1D_FLUSH is available to clean L1D cache (L1D_FLUSH)
rdctl-no	Processor is not susceptible to Rogue Data Cache Load (RDCL)
skip-l1dfl-vmentry	Skip L1D_FLUSH whenever there is a VM_ENTRY

Check The Linux Kernel user's and administrator's guide to understand how which CPUs have to have SMT disabled to guarantee they're not vulnerable, and why SMT mitigation can't be fully guaranteed when inside a virtual machine. The kernel command line allows to control the MDS mitigations at boot time with the option mds=(full|full,nosmt|off).

QEMU/KVM CPUs Mitigation Flags (New CPUs, Microcode) and HOWTO Enable Flags inside Guests

1. Side Channel Attacks - Spectre and Meltdown

amd64

Generally, for Intel CPUs with updated microcode, spec-ctrl will enable both IBRS and IBPB functionality. For AMD EPYC processors, ibpb can be used to enable IBPB specifically, and is thought to be sufficient by itself for that particular architecture.

These flags can be set in a similar manner as other CPU flags, i.e.:

```
qemu-system-x86_64 -cpu qemu64,+spec-ctrl,... ...
qemu-system-x86_64 -cpu IvyBridge,+spec-ctrl,... ...
qemu-system-x86_64 -cpu EPYC,+ibpb,... ...
etc...
```

Additionally, for management stacks that lack support for setting specific CPU flags, a set of new CPU types have been added which enable the appropriate CPU flags automatically:

```
qemu-system-x86_64 -cpu Nehalem-IBRS ...
qemu-system-x86_64 -cpu Westmere-IBRS ...
qemu-system-x86_64 -cpu SandyBridge-IBRS ...
qemu-system-x86_64 -cpu IvyBridge-IBRS ...
qemu-system-x86_64 -cpu Haswell-IBRS ...
qemu-system-x86_64 -cpu Haswell-noTSX-IBRS ...
qemu-system-x86_64 -cpu Broadwell-IBRS ...
qemu-system-x86_64 -cpu Broadwell-noTSX-IBRS ...
qemu-system-x86_64 -cpu Skylake-Client-IBRS ...
qemu-system-x86_64 -cpu Skylake-Server-IBRS ...
qemu-system-x86_64 -cpu EPYC-IBPB ...
```

With regard to migration compatibility, spec-ctrl/ibrs (or the corresponding CPU type) should be set the same on both source/target to maintain compatibility. Thus, guests will need to be rebooted to make use of the new features.

pseries

All three options, cap-cfpc, cap-sbbc, and cap-ibs default to “broken” to maintain compatibility with previous versions of QEMU and unpatched host kernels. To enable them you must start QEMU with the desired mitigation strategy specified explicitly. For example:

```
qemu-system-ppc64 ... \
  -machine pseries-2.11,cap-cfpc=workaround,cap-sbbc=workaround,cap-ibs=fixed
```

With regard to migration compatibility, setting any of these features to a value other than the default, “broken”, will require an identical setting (CPU capability/flag) for that option on the source/destination guest. To enable these settings your guests will need to be rebooted at some point.

s390x

Both bpb and ppa15 are enabled by default when using “-cpu host” and when the host kernels supports these facilities. For other CPU models, the flags have to be set manually. For example:

```
qemu-system-s390x -M s390-ccw-virtio-2.11 ... \
-cpu zEC12,bpb=on,ppa15=on
```

With regard to migration, enabling bpb or ppa15 feature flags requires that the source/target also has those flags enabled. Since this is enabled by default for '-cpu host' (when available on the host), you must ensure that **bpb=off,ppa15=off** is used if you wish to maintain migration compatibility with existing guests when using '-cpu host', or take steps to reboot guests with bpb/ppa15 enabled prior to migration.

2. Side Channel Attacks - Others

a) CVE-2018-3665 - Lazy FP Save/Restore (LazyFP)
b) CVE-2018-3693 - Bounds Check Bypass Store (Variant (or Spectre) 1.1 and 1.2 / BCBS)
c) CVE-2018-3640 - Rogue System Register Read (RSRE / Variant 3a)
d) CVE-2018-3639 - Speculative Store Bypass (SSB / Variant 4 / Spectre-NG)

Check if xsaveopt CPU feature is available for the vCPUs, which would be already enough not to suffer from this vulnerability.

SSBD prevents a load from executing speculatively until the address of all older stores are known. The CPU feature ssbd should be advertised to virtual CPUs so Virtual Machine Guests know that disabling Speculative Store Bypass (SSB) is possible.

Like explained in the previous session of this document, SSBD can be enabled/disabled through seccomp or prctl for OS processes, and VM guests. By having the ssbd capability in its vCPUs, will be also able to enable/disable SSB.

- CPU flags to advertise CPUs (or Microcodes) have mitigations

Intel x86 hosts

```
qemu-system-x86_64 -cpu qemu64,+ssbd,... ...
qemu-system-x86_64 -cpu IvyBridge,+ssbd,... ...
```

AMD x86 hosts

```
qemu-system-x86_64 -cpu qemu64,+virt-ssbd,... ...
qemu-system-x86_64 -cpu qemu64,+virt-ssbd,+amd-ssbd... ...
qemu-system-x86_64 -cpu qemu64,+amd-no-ssb,... ...
```

3. L1 Terminal Fault (L1TF)

a) CVE-2018-3615 - Intel SGX (Software Guard Extensions) (Foreshadow / L1TF)
b) CVE-2018-3620 - Operating Systems and System Management Mode (Fault-OS / SMM) (L1TF)
c) CVE-2018-3646 - Virtualization Extensions (L1TF)

Intel x86 hosts

rdctl-no	Processor is not susceptible to Rogue Data Cache Load (RDCL)
skip-l1dfl-vmentry	Skip L1D_FLUSH whenever there is a VM_ENTRY

Both CPU features come from a another CPU feature available in newer CPUs called IA32_ARCH_CAPABILITIES, which is explained in the beginning of this topic ("**QEMU/KVM Mitigations and CPU Flags**"), responsible to enumerate HW mitigations available in newer CPUs. If they're present in your CPU, you should definitely pass them to the Virtual Machine Guests, but always thinking about live migrations among Hosts with different CPU types (and available CPU features).

```
qemu-system-x86_64 -cpu qemu64,+arch-capabilities,+rdctl-no,... ...
qemu-system-x86_64 -cpu qemu64,+arch-capabilities,+skip-l1dfl-vmentry... ...
qemu-system-x86_64 -cpu qemu64,+arch-capabilities,+rdctl-no,+skip-l1dfl-vmentry,... ...
```

4. Microarchitectural Data Sampling (MDS)

a) CVE-2018-12126 - Microarchitectural Store Buffer Data Sampling (MSBDS / Fallout)
b) CVE-2018-12127 - Microarchitectural Load Port Data Sampling (MLPDS / RIDL)
c) CVE-2018-12130 - Microarchitectural Fill Buffer Data Sampling (MFBDS / ZombieLoad)
d) CVE-2019-11091 - Microarchitectural Data Sampling Uncacheable Memory (MDSUM)

Intel x86 hosts

mds-no	Processor is not susceptible Microarch. Data Sampling (MDS)
md-clear	L1D_FLUSH is available to clean L1D cache (L1D_FLUSH)
rdctl-no	Processor is not susceptible to Rogue Data Cache Load (RDCL)
skip-l1dfl-vmentry	Skip L1D_FLUSH whenever there is a VM_ENTRY

All the features bellow can be passed to the virtual machine guest vCPUs in order to inform about host's HW features for mitigations if not using "cpu mode='host-passthrough'". Together with L1DFL HW mitigations, MDS HW mitigations are also informed through the IA32_ARCH_CAPABILITIES feature.

```
qemu-system-x86_64 -cpu qemu64,+arch-capabilities,+mds-no,... ...
qemu-system-x86_64 -cpu qemu64,+arch-capabilities,+md-clear... ...
qemu-system-x86_64 -cpu qemu64,+arch-capabilities,+rdctl-no,+skip-l1dfl-vmentry,... ...
```

< TODO: FINISH LIBVIRT XML EXAMPLES >

Libvirt examples for Ubuntu Bionic and Disco

<qemu:arg value='Cascadelake-Server,ss=on,vmx=on,hypervisor=on,tsc_adjust=on,umip=on,pku=on,md-clear=on,stibp=on,arch-capabilities=on,xsaves=on,invtsc=on,rdctl-no=on,ibrs-all=on,skip-l1dfl-vmentry=on,mds-no=on'/>

Libvirt examples for Ubuntu Eoan and Focal

Spectre and Meltdown mitigation detection tool

There are different tools that can be used to detect these vulnerabilities in an environment (Architecture, CPU, Firmware Levels, OS Kernel, Hypervisor Types and Versions) AND to show you if all of them are mitigated or not.

A very good one that can be used is: https://github.com/speed47/spectre-meltdown-checker

Bellow you will find this tools output from a Cascade Lake CPU Server. Note that this document tried to explain all its acronyms for CPU Vulnerabilities and/or Mitigation Techniques above, so do search this document for an acronym found at the tool output if you need further explanation.

TIP: I'd run the tool in the Hypervisor Host AND inside the VM OS, to make sure that both, the Host and the Guest have all mitigations in place. With the information provided in previous section of this document (QEMU/KVM Guests HW Mitigations) you will be able to make sure you're correctly advertising your REAL CPU Mitigation Flags into your Guests (so their OS can choose better technique for each of the vulnerabilities).

Spectre and Meltdown mitigation detection tool v0.42-1-g91d0699

Checking for vulnerabilities on current system Kernel is Linux
4.18.0-23-generic #24~18.04.1-Ubuntu SMP Thu Jun 13 17:08:52 UTC 2019 x86_64
CPU is Intel(R) Xeon(R) Gold 6252 CPU @ 2.10GHz

Hardware check

```
* Hardware support (CPU microcode) for mitigation techniques

  * Indirect Branch Restricted Speculation (IBRS)
    * SPEC_CTRL MSR is available: YES
    * CPU indicates IBRS capability: YES (SPEC_CTRL feature bit)

  * Indirect Branch Prediction Barrier (IBPB)
    * PRED_CMD MSR is available: YES
    * CPU indicates IBPB capability: YES (SPEC_CTRL feature bit)

  * Single Thread Indirect Branch Predictors (STIBP)
    * SPEC_CTRL MSR is available: YES
    * CPU indicates STIBP capability: YES (Intel STIBP feature bit)

  * Speculative Store Bypass Disable (SSBD)
    * CPU indicates SSBD capability: YES (Intel SSBD)

  * L1 data cache invalidation
    * FLUSH_CMD MSR is available: YES
    * CPU indicates L1D flush capability: YES (L1D flush feature bit)

  * Microarchitecture Data Sampling
    * VERW instruction is available: YES (MD_CLEAR feature bit)

  * Enhanced IBRS (IBRS_ALL)
    * CPU indicates ARCH_CAPABILITIES MSR availability: YES
    * ARCH_CAPABILITIES MSR advertises IBRS_ALL capability: YES

  * CPU explicitly indicates not being vulnerable to Meltdown/L1TF (RDCL_NO): YES

  * CPU explicitly indicates not being vulnerable to Variant 4 (SSB_NO): NO

  * CPU/Hypervisor indicates L1D flushing is not necessary on this system: YES

  * Hypervisor indicates host CPU might be vulnerable to RSB underflow (RSBA): NO

  * CPU explicitly indicates not being vulnerable to Microarchitectural Data Sampling (MDS_NO): YES

  * CPU supports Software Guard Extensions (SGX): NO

  * CPU microcode is known to cause stability problems: NO
    (model 0x55 family 0x6 stepping 0x6 ucode 0x4000021 cpuid 0x50656)

  * CPU microcode is the latest known available version: NO
    (latest version is 0x4000024 dated 2019/04/07 according to builtin MCExtractor DB v112 - 2019/05/22)
```

CPU vulnerability to the speculative execution attack variants

```
  * Vulnerable to CVE-2017-5753 (Spectre Variant 1, bounds check bypass): YES
  * Vulnerable to CVE-2017-5715 (Spectre Variant 2, branch target injection): YES
  * Vulnerable to CVE-2017-5754 (Variant 3, Meltdown, rogue data cache load): NO
  * Vulnerable to CVE-2018-3640 (Variant 3a, rogue system register read): YES
  * Vulnerable to CVE-2018-3639 (Variant 4, speculative store bypass): YES
  * Vulnerable to CVE-2018-3615 (Foreshadow (SGX), L1 terminal fault): NO
  * Vulnerable to CVE-2018-3620 (Foreshadow-NG (OS), L1 terminal fault): NO
  * Vulnerable to CVE-2018-3646 (Foreshadow-NG (VMM), L1 terminal fault): NO
  * Vulnerable to CVE-2018-12126 (Fallout, microarchitectural store buffer data sampling (MSBDS)): NO
  * Vulnerable to CVE-2018-12130 (ZombieLoad, microarchitectural fill buffer data sampling (MFBDS)): NO
  * Vulnerable to CVE-2018-12127 (RIDL, microarchitectural load port data sampling (MLPDS)): NO
  * Vulnerable to CVE-2019-11091 (RIDL, microarchitectural data sampling uncacheable memory (MDSUM)): NO
```

CVE-2017-5753 aka 'Spectre Variant 1, bounds check bypass'

```
* Mitigated according to the /sys interface: YES (Mitigation: __user pointer sanitization)
* Kernel has array_index_mask_nospec: YES (1 occurrence found of x86 64 bits array_index_mask_nospec())
* Kernel has the Red Hat/Ubuntu patch: NO
* Kernel has mask_nospec64 (arm64): NO

> STATUS: NOT VULNERABLE (Mitigation: __user pointer sanitization)
```

CVE-2017-5715 aka 'Spectre Variant 2, branch target injection'

```
* Mitigated according to the /sys interface: YES
 (Mitigation: Enhanced IBRS, IBPB: conditional, RSB filling)

* Mitigation 1
  * Kernel is compiled with IBRS support: YES
    * IBRS enabled and active: YES
  * Kernel is compiled with IBPB support: YES
    * IBPB enabled and active: YES

* Mitigation 2
  * Kernel has branch predictor hardening (arm): NO
  * Kernel compiled with retpoline option: YES
  * Kernel supports RSB filling: YES

> STATUS: NOT VULNERABLE (IBRS + IBPB are mitigating the vulnerability)
```

CVE-2017-5754 aka 'Variant 3, Meltdown, rogue data cache load'

```
* Mitigated according to the /sys interface: YES (Not affected)
* Kernel supports Page Table Isolation (PTI): YES
  * PTI enabled and active: NO
  * Reduced performance impact of PTI: YES
    (CPU supports INVPCID, performance impact of PTI will be greatly reduced)
* Running as a Xen PV DomU: NO

> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)
```

CVE-2018-3640 aka 'Variant 3a, rogue system register read'

```
* CPU microcode mitigates the vulnerability: YES

> STATUS: NOT VULNERABLE (your CPU microcode mitigates the vulnerability)
```

CVE-2018-3639 aka 'Variant 4, speculative store bypass'

```
* Mitigated according to the /sys interface: YES
  (Mitigation: Speculative Store Bypass disabled via prctl and seccomp)
* Kernel supports disabling speculative store bypass (SSB): YES (found in /proc/self/status)
* SSB mitigation is enabled and active: YES (per-thread through prctl)
* SSB mitigation currently active for selected processes: YES
  (systemd-journald systemd-logind systemd-networkd systemd-resolved systemd-timesyncd systemd-udevd)

> STATUS: NOT VULNERABLE (Mitigation: Speculative Store Bypass disabled via prctl and seccomp)
```

CVE-2018-3615 aka 'Foreshadow (SGX), L1 terminal fault'

```
* CPU microcode mitigates the vulnerability: N/A

> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)
```

CVE-2018-3620 aka 'Foreshadow-NG (OS), L1 terminal fault'

```
* Mitigated according to the /sys interface: YES (Not affected)
* Kernel supports PTE inversion: YES (found in kernel image)
* PTE inversion enabled and active: NO

> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)
```

CVE-2018-3646 aka 'Foreshadow-NG (VMM), L1 terminal fault'

```
* Information from the /sys interface: Not affected
* This system is a host running a hypervisor: YES
* Mitigation 1 (KVM)
  * EPT is disabled: NO
* Mitigation 2
  * L1D flush is supported by kernel: YES (found flush_l1d in /proc/cpuinfo)
  * L1D flush enabled: NO
  * Hardware-backed L1D flush supported: YES (performance impact of the mitigation will be greatly reduced)
  * Hyper-Threading (SMT) is enabled: YES

> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)
```

CVE-2018-12126 aka 'Fallout, microarchitectural store buffer data sampling (MSBDS)'

```
* Mitigated according to the /sys interface: YES (Not affected)
* Kernel supports using MD_CLEAR mitigation: YES (md_clear found in /proc/cpuinfo)
* Kernel mitigation is enabled and active: NO
* SMT is either mitigated or disabled: NO

> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)
```

CVE-2018-12130 aka 'ZombieLoad, microarchitectural fill buffer data sampling (MFBDS)'

```
* Mitigated according to the /sys interface: YES (Not affected)
* Kernel supports using MD_CLEAR mitigation: YES (md_clear found in /proc/cpuinfo)
* Kernel mitigation is enabled and active: NO
* SMT is either mitigated or disabled: NO

> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)
```

CVE-2018-12127 aka 'RIDL, microarchitectural load port data sampling (MLPDS)'

```
* Mitigated according to the /sys interface: YES (Not affected)
* Kernel supports using MD_CLEAR mitigation: YES (md_clear found in /proc/cpuinfo)
* Kernel mitigation is enabled and active: NO
* SMT is either mitigated or disabled: NO

> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)
```

CVE-2019-11091 aka 'RIDL, microarchitectural data sampling uncacheable memory (MDSUM)'

```
* Mitigated according to the /sys interface: YES (Not affected)
* Kernel supports using MD_CLEAR mitigation: YES (md_clear found in /proc/cpuinfo)
* Kernel mitigation is enabled and active: NO
* SMT is either mitigated or disabled: NO

> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)
```