SCRUTINY ON MELTDOWN AND SPECTRE

B.Anirudh Srinivas
Sri Sai Ram Institute Of Technology
Tamil Nadu, India

H.Vidhya
Sri Sai Ram Engineering College
Tamil Nadu, India

Dr.S.Rajarajan
Sri Sai Ram Institute Of Technology
Tamil Nadu, India

Abstract: Meltdown and Spectre are two major micro-architectural threats in modern processors that exploit imperative techniques such as speculative execution and out-of-order execution to enhance processor performance. In both cases, classified information is obtained by malicious sources via side-channel attacks, a cataclysmic consequence. The Common Vulnerabilities and Exposures IDs denoting spectre are CVE-2017-5753 for Bounds Check Bypass, CVE-2017-5715 for Branch Target Injection and for meltdown it is CVE-2017-5754 for Rogue Data Cache Load. While patches are being developed by companies to address the challenge, a detailed account of the security threats is rather eluding the scientific community. This paper will survey the detailed consequences of the spectre and meltdown threats and the mitigation approaches proposed in chronological order. This will serve as an edifice to understand the current security vulnerabilities and help to forestall future threats.

Index terms – Hardware security, Computer architecture, Speculative execution, Out-Of-Order-execution.

I. INTRODUCTION

In the world of computer security, the embezzlement of sensitive data has always been addressed as a major issue. Users are bound to access the internet everywhere for their needs. Exploitation of data has reached its next level in which the hackers are now able to access raw information of users by speculative execution and Out-Of-Order execution.

Speculative execution is an enhanced technique where a computer performs some task that may not be needed. Work is done ahead before it is known whether it is actually needed, so as to prevent a delay that would have to be provoked by doing the work after it is known that it is needed. If it turns out the work was not needed, after all, most changes made by the work are reverted and the results are ignored. The objective is to provide more concurrency if extra resources are available. This approach is employed in a wide range of areas, including branch prediction in pipelined processors, value prediction for exploiting value locality, prefetching memory and files. Branch Misprediction leads to leakage of sensitive information.

Out-of-order execution is an approach that is used in high-performance processors. This approach efficiently uses instruction cycles (is a process by which a computer retrieves program instruction from its memory, determines what action the instruction requires and carries out those actions.) and reduces costly delay. A processor will execute the instructions in an order of availability of data or operands instead of the original order of the instructions in the program. The processor will avoid being idle while data is retrieved for the next instruction in a program.

In other words, a processor that uses multiple execution units completes the processing of instructions in the wrong order. For example, I1 and I2 are the two instructions where I1 comes first then I2. In the out-of-order execution, a processor can execute I2 instruction before I1 instruction has been completed. This flexibility will improve the performance of the processor since it allows execution with less waiting time.

With the help of these two important phenomena, Meltdown and spectre occur in modern processors. In this paper, we present detailed scrutiny of these two attacks, countermeasures which are proposed by various multinational companies to curb the attack.

II. LITERATURE SURVEY

A) Histories of the attack

The framework that gave rise to meltdown and spectre attack was when a paper called “The Intel 80x86 Processor Architecture: Pitfalls for Secure Systems" published at the 1995 IEEE Symposium on Security and Privacy warned against a covert timing channel in the CPU cache and translation lookaside buffer (TLB). This analysis was performed under the auspices of the National Security Agency’s Trusted Products Evaluation Program (TPEP). Operating systems like Linux started to realize this issue and went on working to mitigate the address leaks. On the other hand, Apple’s XNU kernel adopted KASLR (Kernel address space layout randomization) in its new version of OS called OS X Mountain lion 10.8. Researches of TU Graz presented a paper on side channels attacks and ARM Cache Attacks on Mobile Devices which was found to be a substantial hassle. Abasement of KASLR on cache-based architectures was put forth by Bosman from Vrije University.

In June 2017, KASLR was found to have a large class of new vulnerabilities. Research at the Graz University of Technology showed how to solve these vulnerabilities by preventing all access to unauthorized pages. A presentation on the resulting KAISER

technique was submitted for the Black Hat congress in July 2017 but was rejected by the organizers. Nevertheless, this work led to kernel page-table isolation (KPTI, originally known as KAISER) in 2017, which was confirmed to eliminate a large class of security bugs, including the not-yet-discovered Meltdown – a fact confirmed by the Meltdown authors.

In July 2017, research was made public on the CyberWTF website by security researcher Anders Fogh outlined the use of a cache timing attack to read kernel space data by observing the results of speculative operations conditioned on data fetched with invalid privileges.

The Meltdown was discovered independently by Jann Horn from Google's Project Zero, Werner Haas and Thomas Prescher from Cyberus Technology, as well as Daniel Gruss, Moritz Lipp, Stefan Mangard and Michael Schwarz from the Graz University of Technology. An Another CPU vulnerability was put forth by Jann Horn from Google's Project Zero along with Paul Kocher, Daniel Genkin, Mike Hamburg, Moritz Lipp, Yuval Yarom and Data61 which is called as Spectre.

B) Advancements in the attack

After the previous attacks were being reported by Intel, AMD and ARM processors, eight new security flaws rose on 21st May 2018, which was named as spectre NG. Spectre NG was again reported by Intel, AMD and ARM processors. These new set of variants were described as Rogue system register read (CVE-2018-3640), Speculative store bypass(CVE-2018-3639) which was named as spectre variant 3a and variant 4 respectively by CVE and was posted on NVD webpage. In accordance to these, Amazon Deutschland, Cyberus Technology, SYSGO, and Colin Percival (FreeBSD), Intel has revealed details on the third Spectre-NG variant CVE-2018-3665 (Lazy FP State Restore, Intel SA-00145) on June 13, 2018. It is also known as Lazy FPU state leak (abbreviated "LazyFP") and "Spectre-NG 3". On July 10, 2018, Intel revealed details on another Spectre-NG class vulnerability called "Bounds Check Bypass Store"(BCBS), aka"Spectre 1.1" (CVE-2018-3693), which was able to write as well as read out of bounds. Another variant named "Spectre 1.2" was mentioned as well.

In late July 2018, researchers at the universities of Saarland and California revealed ret2spec (aka "Spectre v5") and SpectreRSB, new types of code execution vulnerabilities using the Return Stack Buffer (RSB).

At the end of July 2018, researchers at the University of Graz revealed "NetSpectre", a new type of remote attack similar to Spectre V1, but which does not need attacker-controlled code to be run on the target device at all.

Thus, as months are being passed by many companies are indagating new types of flaws that could emerge from processors which are associated with these two flaws. However, Intel said that it would manufacture new chips, provide efficient software patches that could curb the series of meltdown and spectre attacks.

As stratification of attacks is rising, Major companies like Intel, Microsoft are grappling hard to manufacture chips and create software patches that could eradicate the problem.

III) STRATIFICATION OF ATTACKS

Spectre and meltdown can be predominantly stratified as follows:

i) Bounds check bypass (CVE-2017-5753)

As mentioned, speculative execution is a technique where the flow of the programs is speculated by the processor even before necessary commands are given. In case the decision was right, performance is improved by saving up on idle time. Otherwise, the results of the speculation obtained must be sent to the right path and the wrong values are discarded. At the program level, this process is invisible but due to speculative execution, there may be certain hints which can be of help to a malicious source for exploiting memory. Using bounds check bypass, malicious sources can access data that has been used in speculative execution. One sub-variant of this technique is bounds check bypass store, which uses the speculative store to overwrite a smaller speculative load that creates a side-channel which is handled by malicious hackers.

//SNIPPET CODE FOR BOUNDS CHECK BYPASS

```
char flag;
char cmd[BUFFER_LENGTH]
&lt;snip&gt;
if (pos &gt;= BUFFER_LENGTH)
return -1;
cmd[pos++] = tok;
If (flag == 'R')
read_cmd(cmd)
else
write_cmd(cmd)
```

In the above example, while the processor is waiting (for example for a slow load to complete from DDR memory) to confirm if pos exceeds the buffer length or not, it could speculatively execute the instruction which updates the cmd buffer. Thus bypassing the bounds check. If the user could control the value of the pos variable, they may be able to speculatively overflow the cmd buffer in such a way as to affect subsequent speculative operations. Speculatively executed instructions are not normally visible to the user or programmer (they are not committed to the architectural program state). When the processor eventually resolves the value of pos and the bounds check fails, the processor rolls back all speculatively executed instructions and resumes execution after the bounds check along the correct path as if the speculation had never occurred. However, before such a rollback, speculatively executed subsequent instructions would use the change made by the cmd[pos++] = tok operation. If pos is out of bounds, it will overwrite beyond cmd[] buffer, possibly also overwriting nearby data in the memory such as the flag variable. This would, in turn, decide the course of the speculative execution and further changes to the cache memory, which can be observed by a malicious user via targeted cache side-channel attack or analysis methods.

ii) Branch target Injection (CVE-2017-5715)

A branch target predictor predicts the target of a conditional/unconditional branch instruction before the target of instruction is computed by the processor. The branch target injection targets the branch predictor. Indirect branches, unlike direct ones, occur when branch destination is not within the instruction itself. The indirect branch predictor makes use of previously executed branches to predict the destination of the current indirect branch. Branch target injection is composed of five elements:

(i)The victim must have some confidential data to be exploited by the malicious code.

(ii)The exploit needs a method to access the secret.

(iii)The exploit's reference should be used when executing the victim's code.

(iv)The indirect branch in the victim's code should speculatively mispredict and execute a gadget.

(v) The gadget must execute during speculation window which closes when the processor determines that gadget execution was incorrect.

//SNIPPET CODE FOR BRANCH TARGET INJECTION

```
class Shape {
public:
virtual void Draw() = 0;
};
class Circle : public Shape {
public:
void Draw() override { … }
};
Where Shape is the base class and Circle is a derived class.
Now consider the following codesegment.
Shape* obj = new Circle;
obj-&gt;Draw();
```

In typical polymorphic code such as the example above, the target address of virtual function Draw() cannot be determined at compile time, thus resulting in an indirect branch that must be resolved at run-time. During run-time, a dynamic lookup is performed to find the matching function. While this happening, the microprocessor guesses the target address and right away starts to speculative execute that code. The attacker needs to find code similar to the example above that when manipulated through the indirect branch predictor, can lead the microprocessor to speculative execute code that results in Spectre Variant 1. The attacker can then use the first variant of the attack to infer sensitive data from the victim's memory space.

iii) Rogue data cache load (CVE-2017-5753)

Meltdown is been performed by a rogue process in which the attacks enforces the CPU to execute an ephemeral instruction which gets all the unauthorised secret data stored in the physical memory. This process is most commonly found in all types of operating systems like Windows, Linux or MAC OS, especially on a 64-bit processor.

i) Firstly, the attacker makes sure that all the information of the memory location is been loaded into a register.

ii) After that, an instruction is been passed dependent on the content of the register.

iii) Then the process executes a timing attack or side channel attack which runs through the cache and the secret information that is been stored at the memory.

This technique is been deployed in meltdown to read every address of interest at high speed depending on alternative processes which contain all the sensitive information from any address that is been memory mapped.

INSTRUCTION SEQUENCE OF MELTDOWN:

```
1; rcx = kernel address
2; rbx = probe array
3retry:
4mov al, byte [rcx]
5shl rax, 0xc
6jz retry
7mov rbx, qword [rbx + rax]
```

iv) Rogue System Register Read (CVE-2018-3640)

Rogue System Register Read, described as Variant 3a uses speculative execution and side channel cache methods to infer the value of some 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 that is 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 states exposed by the Variant 3a method are not secret or sensitive, and does not directly enable attacks or exposure of user data. However, 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. As the rogue system register read method involves attacker-controlled code execution, a local attacker who employs rogue system register read to break KASLR may be low impact for most end users.

v) Speculative Store bypass (CVE-2018-3639)

It takes advantage of speculative execution in a similar way to the Meltdown and Spectre security vulnerabilities. It affects the ARM, AMD and Intel families of processors. It was discovered by researchers at Microsoft Security Response Center and Google Project Zero. After being leaked on May 3 2018, as part of a group of eight additional Spectre-class flaws provisionally named *Spectre-NG*. It was first disclosed to the public as "**Variant 4**" on May 21, 2018, alongside a related speculative execution vulnerability designated "Variant 3a".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 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 environments where the attacker has to discover such a confused deputy instead of being able to cause it to be generated.

vi) Lazy FP State Restore(CVE-2018-3665)

The vulnerability is caused by a combination of flaws in the speculative execution technology present within the affected CPUs and how certain operating systems handle context switching on the floating point unit (FPU). By exploiting this vulnerability, a local process can leak the content of the FPU registers that belong to another process. This vulnerability is related to the Spectre and Meltdown vulnerabilities that were publicly disclosed in January 2018.

It was announced by Intel on 13 June 2018, after being discovered by employees at Amazon, Cyberus Technology and SYSGO. System software may utilize the Lazy FP state restore technique to delay the restoring of state until an instruction operating on that state is actually executed by the new process. Systems using Intel Core-based microprocessors may potentially allow a local process to infer data utilizing Lazy FP state restore from another process through a speculative execution side channel. System software may opt to utilize Lazy FP state restore instead of eager save and restore of the state upon a context switch. Lazy restored states are potentially vulnerable to exploits where one process may infer register values of other processes through a speculative execution side channel that infers their value. This flaw is mainly reported in Intel core-based microprocessors. It is possible to exploit this bug without actually triggering any operating system traps. By placing the FPU access in the shadow of a forced branch misprediction (e.g. using a retpoline) the processor will still speculatively execute the code but will rewind to the mispredicted branch and never actually execute the operating system trap. This allows the attack to be rapidly repeated, quickly reading out the entire FPU and SIMD register state.

vii) Bounds Check Bypass Restore (CVE-2018-3693)

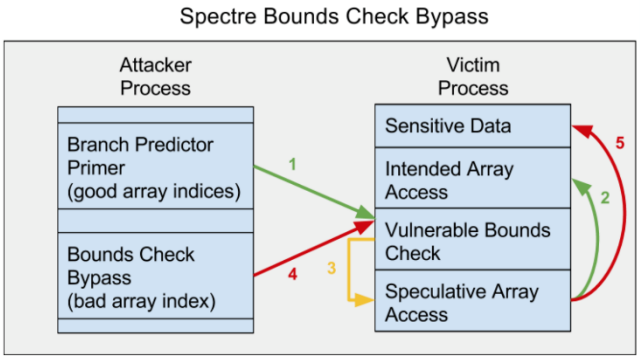
The Speculative Bounds Check Bypass Store variant could be used to influence the course of the speculative execution by targeting local automatic variables or function pointers and/or return addresses found in the memory. It might be used to direct the processor's speculative engine to different code paths, the effects of which are observed by an attack via side channels to know memory contents. This can then be leveraged to access secret information and/or build another attack.

IV) MITIGATIONS OF ATTACKS

Meltdown bug can be alleviated by a KAISER patch otherwise called as kernel address space layout randomization (KPTI table) whereas spectre, in turn, is more difficult to fix than meltdown with researchers predicting that new variants may arise in the future. Patches have been applied for CPU firmware, OS patches and browsers.

a) Bounds check bypass

Software mitigation techniques can serve as a barrier between a bounds check and the future side channel. The LFENCE instruction is one such software mitigation. It does not execute until all the previous instructions have been executed. Also, no other instruction begins to execute until LFENCE instruction is executed. The LFENCE can be possibly placed after range check and branch but before any code that uses the checked value. Other instructions such as CMOVcc, ADC, ADD, SBB and SETcc can also be used as mitigation in six current processors of Intel, but it is not known if they will be effective in future Intel processors.



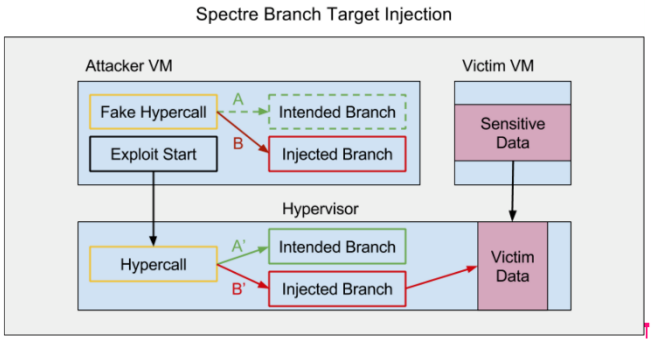
b) Branch target injection

This is even more difficult to mitigate using the software. As discussed, branch target injection aims to influence targets of indirect branches. Either hardware can be manipulated or speculation is controlled indirectly. Mitigation requires either installing a CPU microcode update from the CPU vendor or by applying software mitigation like Google's Retpoline. Retpoline is a binary modification technique that is used against this variant. It may be applied to the kernel, system programs and libraries and also individual programs. Various other patches were released by companies such as Amazon, AMD, ARM, Apple, Google, Microsoft, Cisco, Intel, Linux, Azure, Mozilla, NVidia, Red Hat, Ubuntu, XEN etc. 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.

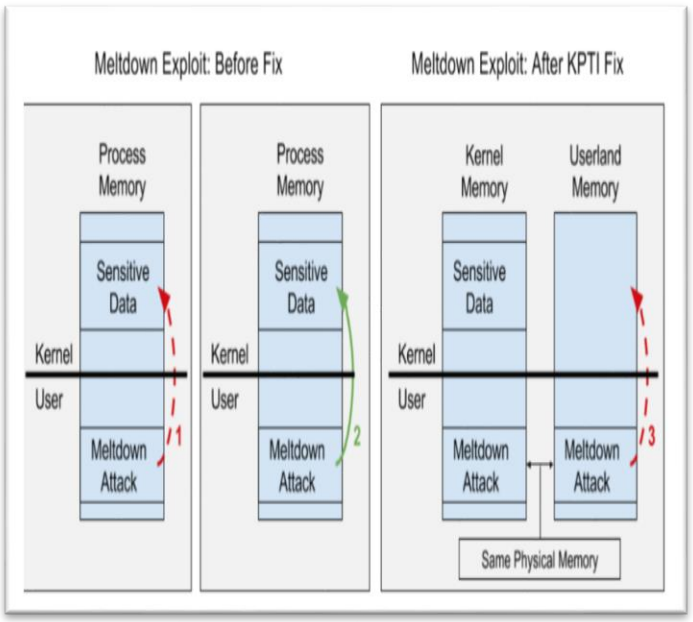
•Single Thread Indirect Branch Predictors (STIBP): Prevents indirect branch predictions from
Being controlled by the sibling Hyperthread.

•Indirect Branch Predictor Barrier (IBPB): Ensures that earlier code's behaviour does not control later indirect branch predictions.



c) Rogue Data Cache Load

This basic dual-page-table approach was previously proposed as 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 the 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.



d) Rogue system register read

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.

e) Speculative store bypass

Intel is planning to address Variant 4 by releasing a microcode patch that creates a new hardware flag named *Speculative Store Bypass Disable (SSBD)*. A stable microcode patch is yet to be delivered, with Intel suggesting that the patch will be ready "in the coming weeks". Many operating system vendors will be releasing software updates to assist with mitigating Variant 4; however, microcode/firmware updates are required for the software updates to have an effect. 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.

f) *Lazy FP State Restore*

For Lazy FP State restore, it is possible to mitigate the vulnerability at the operating system and hypervisor levels by always restoring the FPU state when switching process contexts. With such a fix, no firmware upgrade is required. Some operating systems already did not lazily restore the FPU registers by default, protecting those operating systems on affected hardware platforms, even if the underlying hardware issue existed. On Linux operating system using kernel 3.7 or higher, it is possible to force the kernel to eagerly restore the FPU registers by using the eagerfpu=on kernel parameter. Also, many system software vendors and projects, including Linux distributions, OpenBSD and Xen have released patches to address the vulnerability. Recommended mitigation for Speculative Bounds Check Bypass to ensure that speculative execution does not occur past the bounds check instructions. This can be achieved by introducing serializing instructions like 'lfence' after the bounds check instructions and before the following instructions.

```
if(pos>=BUFFER_LENGTH)

return -1;

asm volatile

("lfence":::"memory");
cmd[pos++] = tok
```

Alternatively, speculative execution may be controlled by constraining the array index. For example, in the kernel, the array_index_nospec() macro can be used:

```
if (pos >= BUFFER_LENGTH)
return -1;
pos=array_index_nospec(pos,BUFFER_LENGTH);
cmd[pos++] = tok
```

This macro ensures that the returned array index is always within the array bounds and that speculative execution cannot access beyond the bounds of the array. A similar approach can be used in user space.

V) CONCLUSION

The purpose of this scrutiny was to view and understand the recent trend of hardware attacks and mitigations provided by companies to impede the attack. Further research and discussions need to be carried out on attacks which influence this similar kind of mechanisms such as “foreshadow” which was discovered recently to exploit sensitive information.

REFERENCES:

[1] https://meltdownattack.com/meltdown.pdf

[2] https://spectreattack.com/spectre.pdf

[3] https://googleprojectzero.blogspot.com/2018/01/reading-privileged-memory-with-side.html

[4] https://software.intel.com/security-software-guidance/api-app/sites/default/files/336983-Intel-Analysis-of-Speculative-Execution-Side-Channels-White-Paper.pdf

[5] https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00145.html

[6] https://access.redhat.com/solutions/3523601

[7] https://en.wikipedia.org/wiki/Meltdown_(security_vulnerability)

[8] https://en.wikipedia.org/wiki/Spectre_(security_vulnerability)

[9] https://www.csoonline.com/article/3247868/vulnerabilities/spectre-and-meltdown-explained-what-they-are-how-they-work-whats-at-risk.html

[10] https://www.redhat.com/en/blog/what-are-meltdown-and-spectre-heres-what-you-need-know

[11] https://www.wired.com/story/speculative-store-bypass-spectre-meltdown-vulnerability/

[12] https://www.firstpost.com › Technology News › News-Analysis

[13] https://www.pcworld.com/article/3245606/security/intel-x86-cpu-kernel-bug-faq-how-it-affects-pc-mac.html

[14] https://www.zdnet.com/article/new-spectre-attack-can-remotely-steal-secrets-researchers-say/