RED HAT SUMMIT

Exploiting modern microarchitectures

Meltdown, Spectre, and other hardware security vulnerabilities in modern processors

Jon Masters - Computer Architect
CRob - Red Hat Product Security
10May2018

Who are we?

JCM

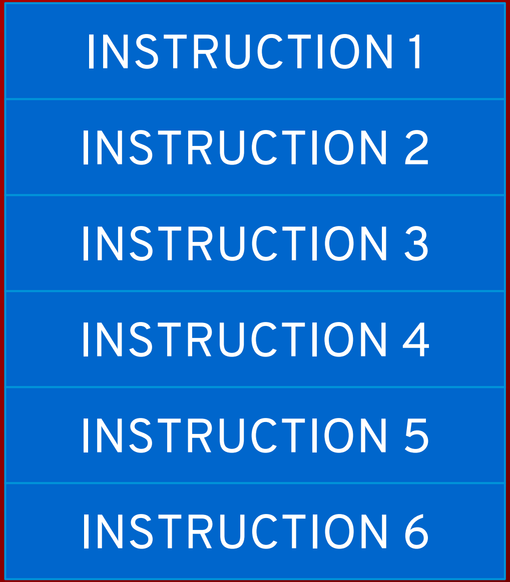
CRob

Out-of-Order (OoO) execution converts sequential “in order” machines (CPUs) as conceptualized by programmers into out-of-order “data flow” machines in which computation is performed as soon as dependent data is available*

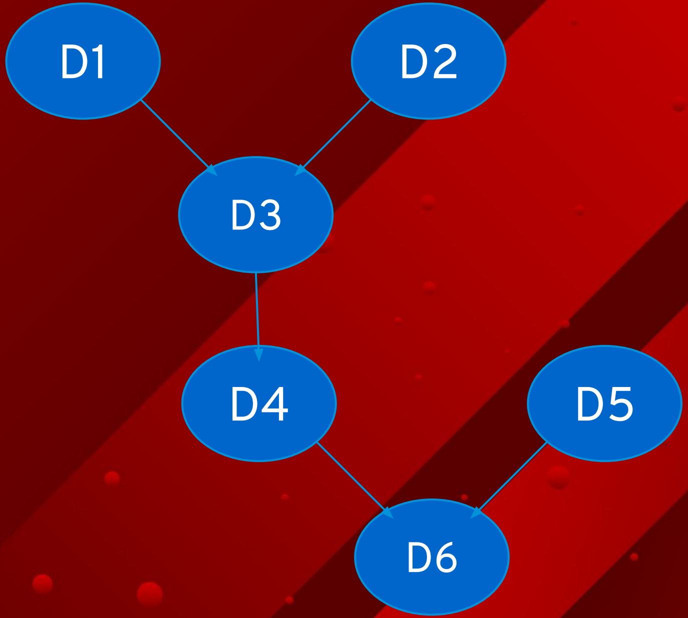
* Instructions issue and retire “in-order”, but are executed using an Out-of-Order “backend”

Sequential vs data flow

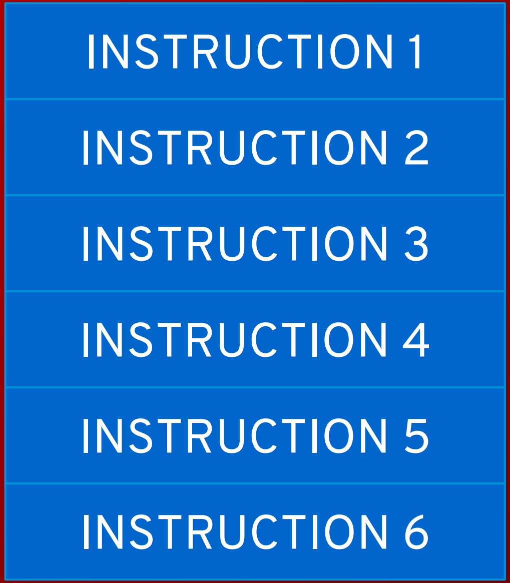
Sequential model



Data flow model



Sequential machines

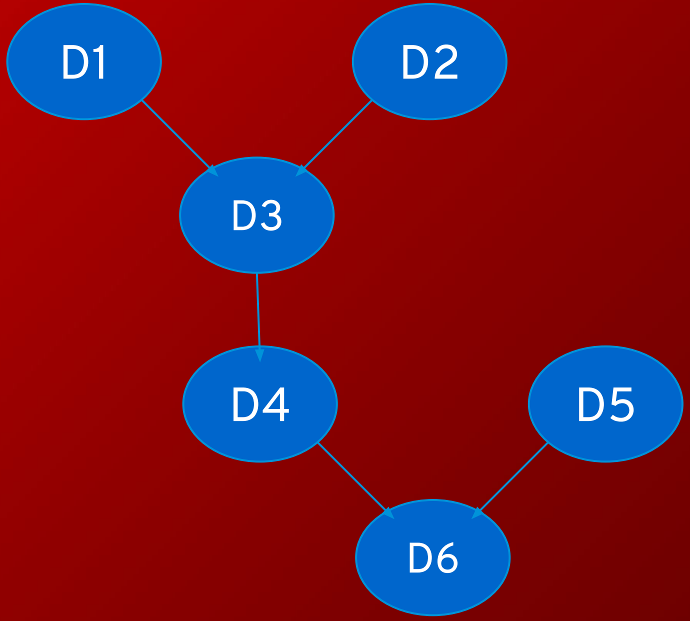


Programmers intuitively think in terms of sequential program flow (“I do this, then I do that”)

Expectation is that the machine does everything in the order written down

Debugging involves tracing what each individual instruction did, one after the other, in the order written

Data flow machines



Data flow operates in terms of data. Each operation proceeds when its dependent data operands are ready

Instructions do not execute in the order written by the programmer

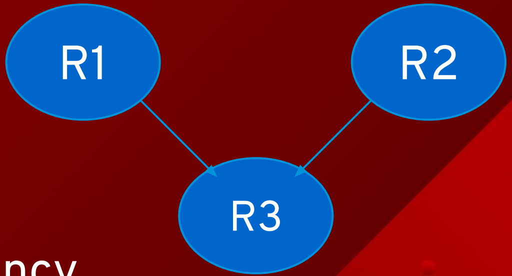
Debugging a real world data flow machine would involve understanding all of the internal machine state

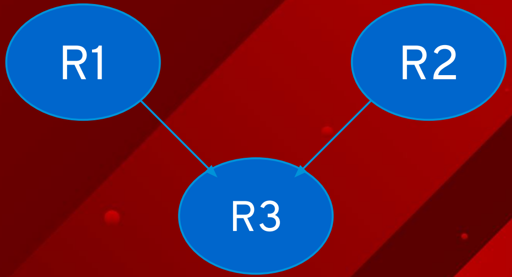
Robert Tomasulo described how to convert an in-order sequential machine (as understood by programmers) into an Out-of-Order data flow model machine

Tomasulo's Algorithm

R1 = LOAD A
R2 = LOAD B
R3 = R1 + R2
R1 = 1
R2 = 1
R3 = R1 + R2

No data dependency


$$\begin{tikzcd} R1 \arrow[dr] & & R2 \arrow[dl] \\ & R3 & \end{tikzcd}$$


$$\begin{tikzcd} R1 \arrow[dr] & & R2 \arrow[dl] \\ & R3 & \end{tikzcd}$$

Tomasulo's Algorithm

R1 = LOAD A
R2 = LOAD B
R3 = R1 + R2
R1 = 1
R2 = 1
R3 = R1 + R2

Program Order

Entry	RegRename	Instruction	Deps	Ready?
1	P1 = R1	P1 = LOAD A	X	Y
2	P2 = R2	P2 = LOAD B	X	Y
3	P3 = R3	P3 = R1 + R2	1,2	N
4	P4 = R1	P4 = 1	X	Y
5	P5 = R2	P5 = 1	X	Y
6	P6 = R3	P6 = P4 + P5	4,5	N

Re-Order Buffer (ROB)

Tomasulo removed the need for chips to wait around for dependent calculations. Deep reorder buffers allow very high performance gains in contemporary servers, laptops, and even phones

…time passed, and computer architects realized it was possible to gain even more performance…

“What if we combine Out-of-Order Execution with Branch Prediction?”

Branches

```
for (i=0;i<10;i++)
{
   // something here
}
// post-loop code here
```

```
          movq $0, %rax
     loop:
          incq %rax
          cmpq $10, %rax
          jle loop
```

```
// pack clothes
if (raining) {
   // pack umbrella
}
// more packing
```

```
                movq $raining, %rax
                cmpq $1, %rax
                jne more_packing
                // pack umbrella
more_packing:
                // more packing
```

Branches

```
// pack clothes
if (raining) {
   // pack umbrella
}
// more packing
```

```
             movq $raining, %rax
             cmpq $1, %rax
             jne more_packing
more_packing:
             ...
```

Branch conditions take time to "resolve" (do I take the branch or not?)

Speculative Execution

R1 = LOAD A
TEST R1
IF R1 ZERO {
R1 = 1
R2 = 1
R3 = R1 + R2

Entry	RegRename	Instruction	Deps	Ready?	Spec?
1	P1 = R1	P1 = LOAD A	X	Y	N
2		TEST R1	1	Y	N
3		IF R1 ZERO {	1	N	N
4	P2 = R1	P4 = 1	X	Y	Y*
5	P3 = R2	P5 = 1	X	Y	Y*
6	P4 = R3	P4 = P2 + P3	4,5	Y	Y*

* Speculatively execute the branch before the condition is known (“resolved”)

Branch prediction allows machines to guess which path code is going to take before that is fully known (“resolved”)

When combined with an Out-of-Order backend, “Speculative Execution” will try to execute code before we know whether we need to do so. If we guess wrong, we throw away our prediction

All is well…unless someone can see what we did speculatively and use it to learn things they should not know…

The Speculation Diner

Look at what we found!

Flash forward nearly 40 years.... 3 groups of researchers independently find....

Speculation is not an unobservable black box. We can abuse speculation to leak information we shouldn’t access using a “side channel”

Side channels are everywhere! And they’re also not new! There are many different mechanisms to discover what a machine is doing through observation

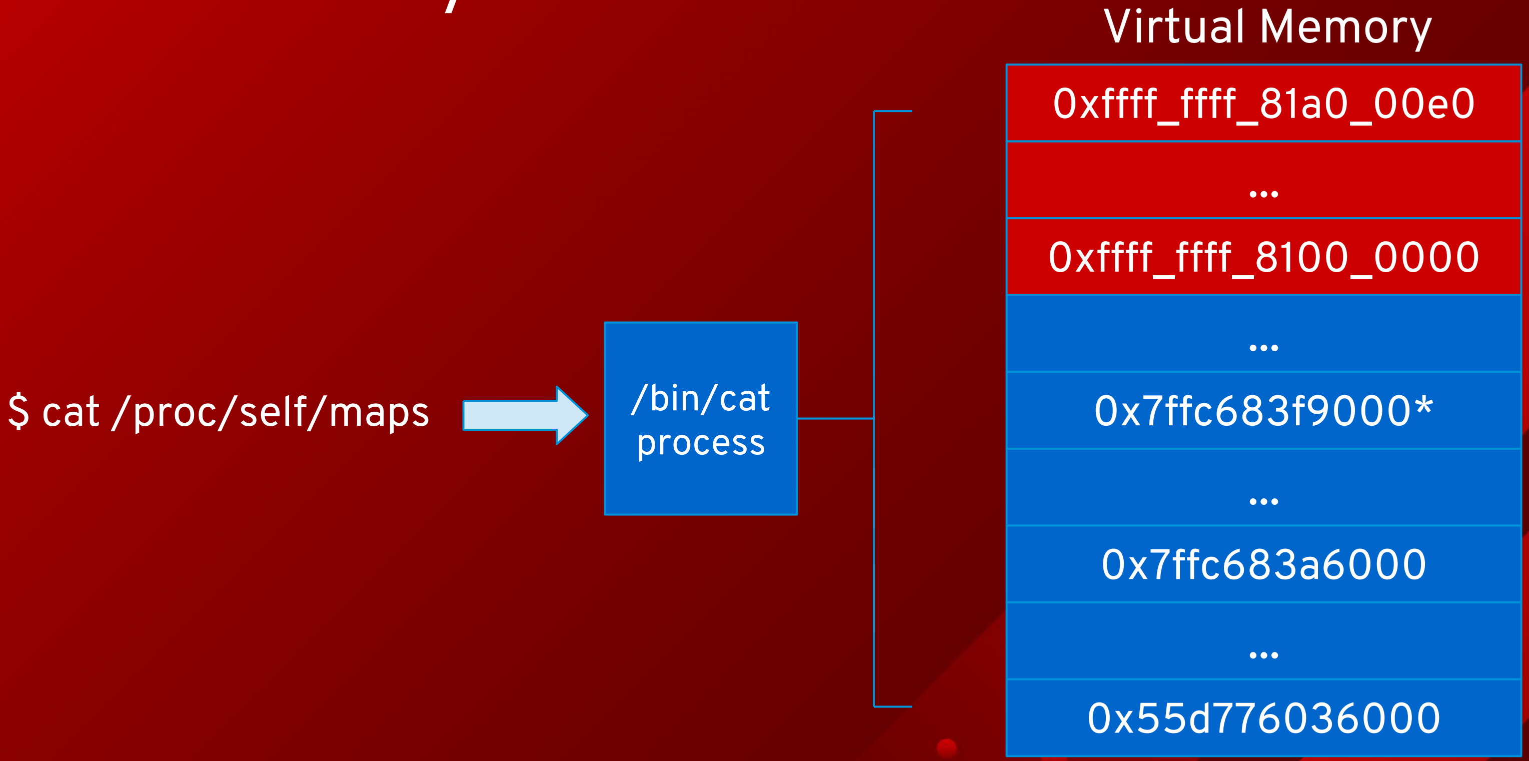
Classic example is “differential power analysis” being used to monitor voltages on the pins of a chip during crypto operations - power varies according to the calculation*

* Your “chip-and-pin” bank card is very carefully designed not to do this

Cache side channels exploit the shared nature of memory on a local system to infer what other programs are doing

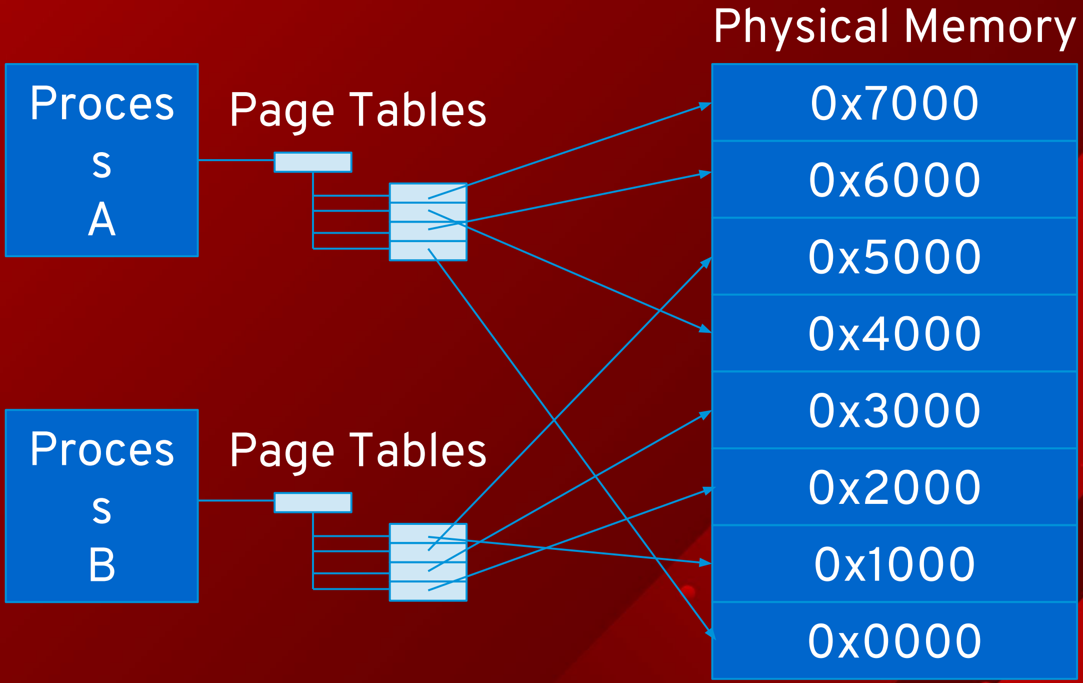
But first…let’s talk about memory…recall how modern running programs use “virtual memory”…

Virtual Memory

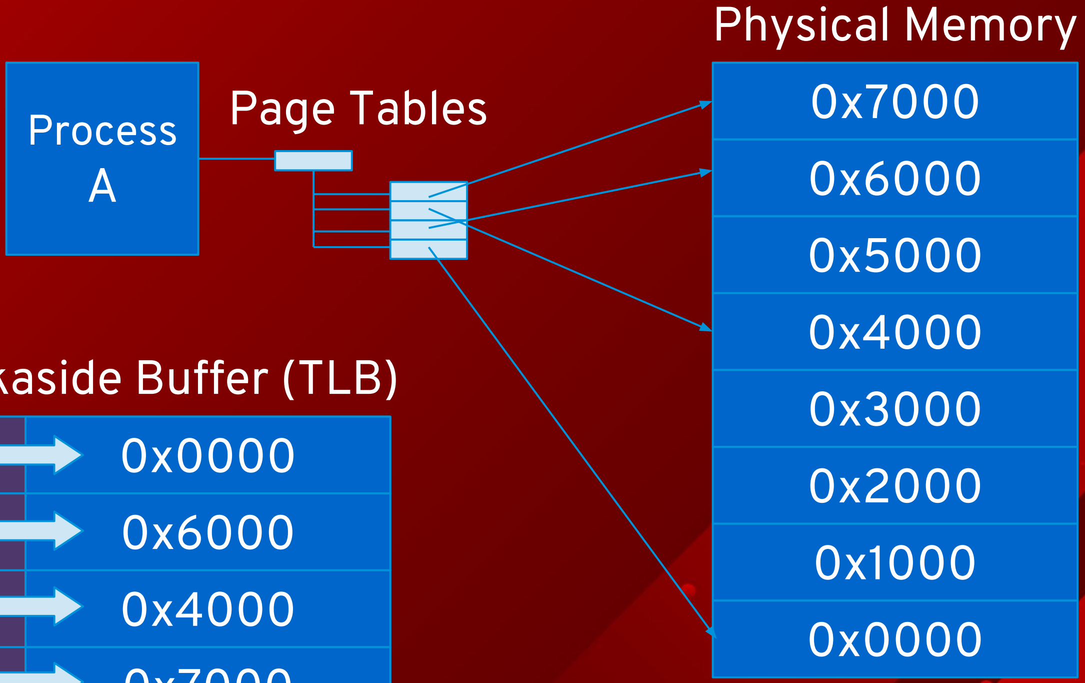


* Special case kernel VDSO (Virtual Dynamic Shared Object)

Virtual Memory



Virtual Memory

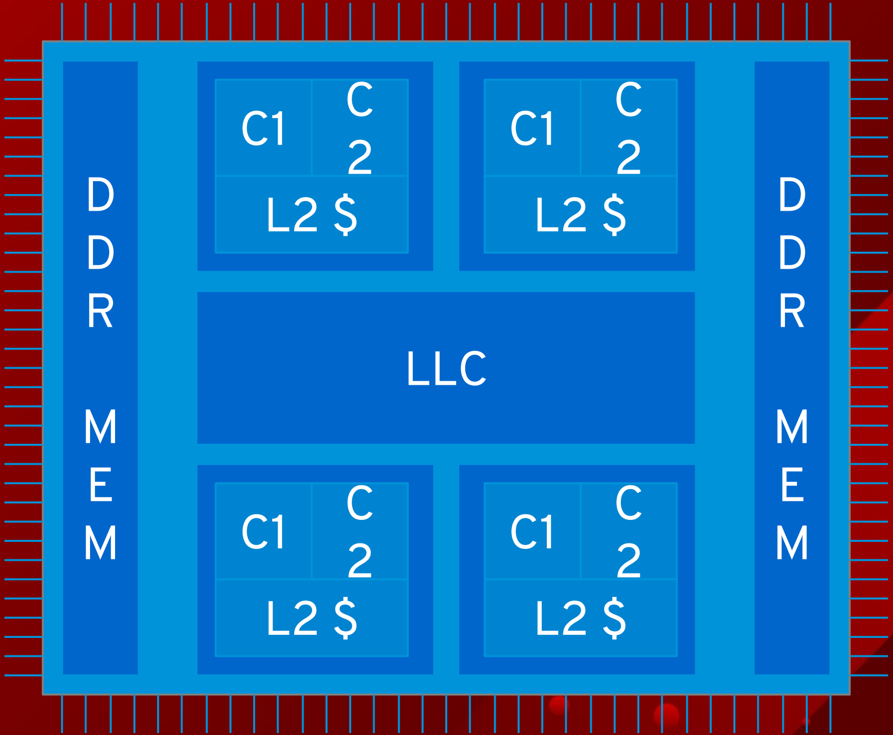


Translation Lookaside Buffer (TLB)

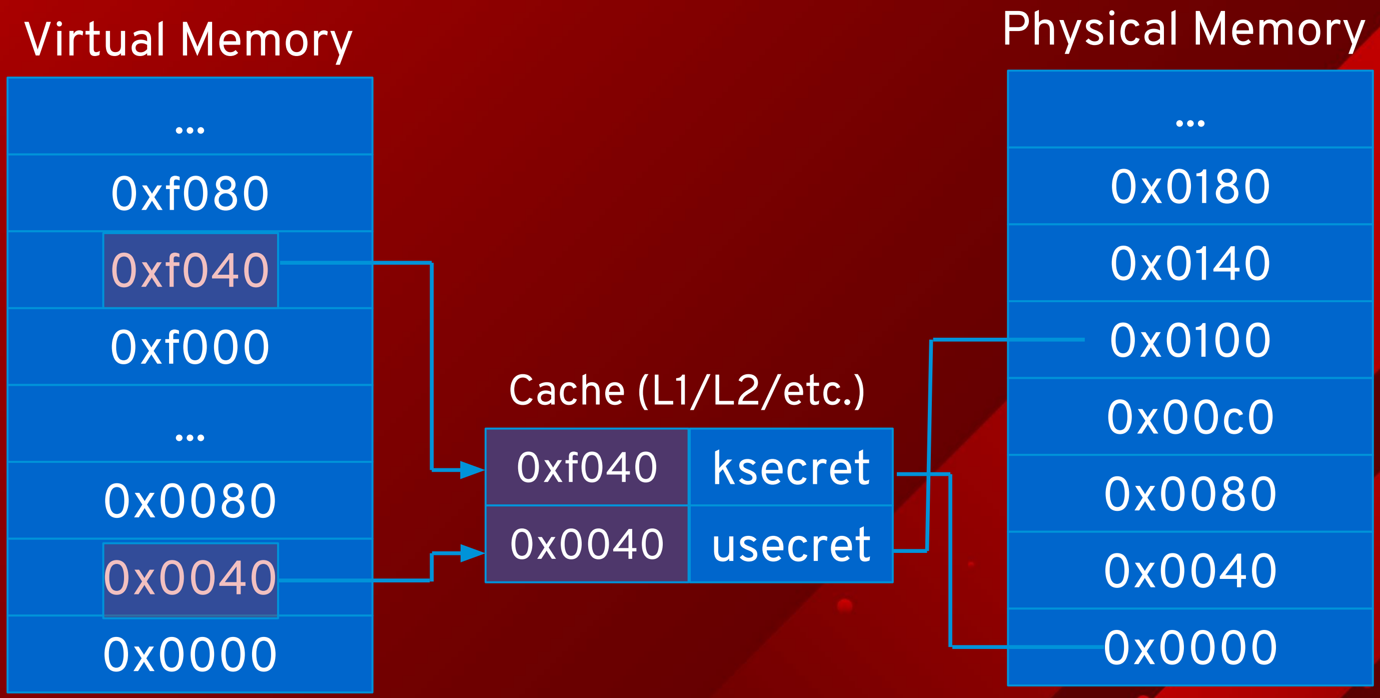
0x4000 →	0x0000
0x3000 →	0x6000
0x1000 →	0x4000
0x0000 →	0x7000

Caches are used to store the actual data read by a running program from this virtual memory

Caches



Caches



* For readability privileged kernel addresses are shortened to begin 0xf instead of 0xffffffff...

Caches are shared resources. The time taken to access data is proportional to whether it is located in the cache

Caches

```
time = rdtsc();
maccess(&data[0x300]);
delta3 = rdtsc() - time;

time = rdtsc();
maccess(&data[0x200]);
delta2 = rdtsc() - time;
```

Execution time taken for instruction is proportional to whether it is in cache(s)

“Meltdown” abuses at-retirement exception (error) handling in the speculation backend to read unauthorized data…

"Meltdown" abuses at-retirement exception (error) handling in the speculation backend to read unauthorized data...

fancy way of saying
"at the very end"

…then uses the cache access trick to reconstruct what that unauthorized data was (hence “side channel”)

We will get a permission check failure but we won’t handle it until too late - after the speculation window completes (we already did the cache access trick)

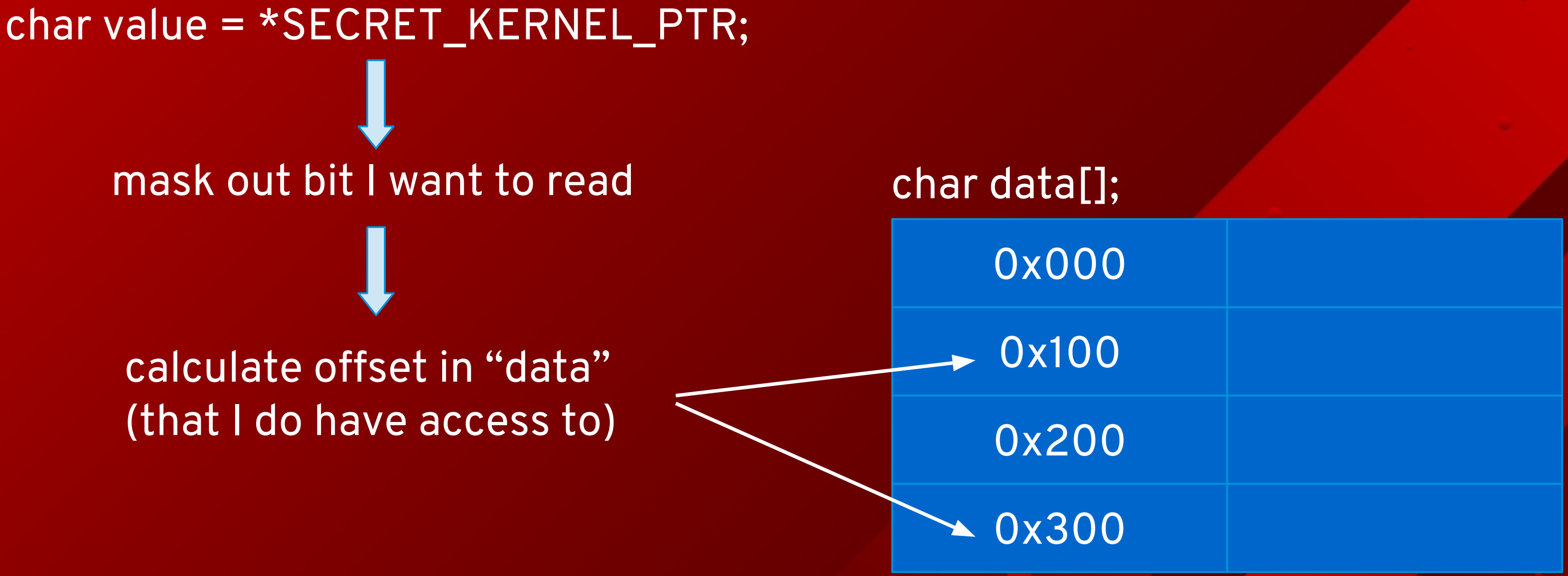
Meltdown

- A malicious attacker arranges for exploit code similar to the following to speculatively execute:

```
if (spec_cond) {
            unsigned char value = *(unsigned char *)ptr;
            unsigned long index2 = (((value>>bit)&1)*0x100)+0x200;
            maccess(&data[index2]);
}
```

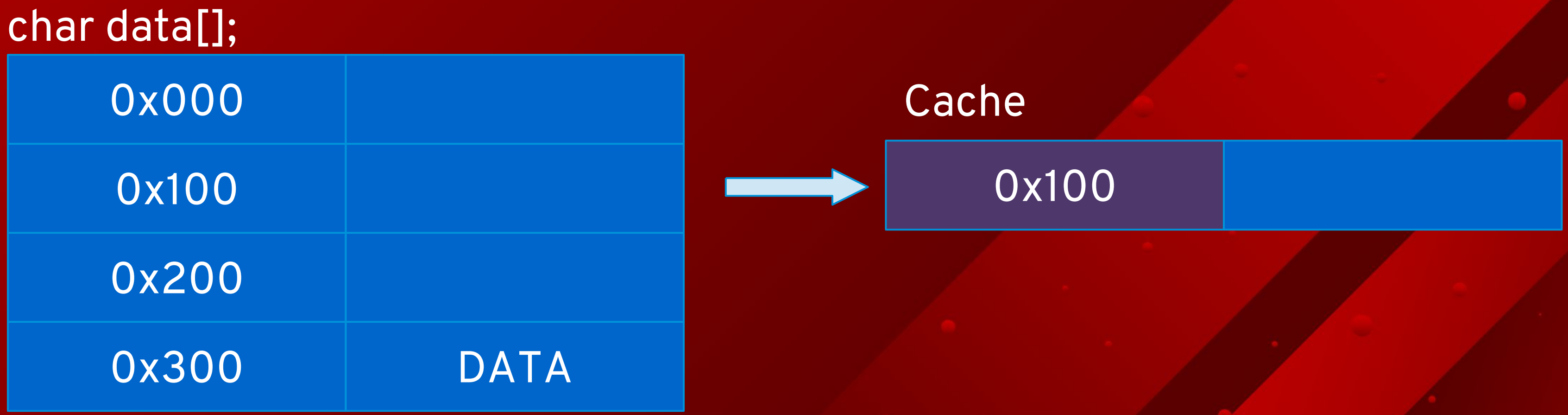
- “data” is a user controller array to which the attacker has access, “ptr” contains privileged data

Meltdown


char value = *SECRET_KERNEL_PTR;
mask out bit I want to read
calculate offset in “data” (that I do have access to)
char data[];
0x000
0x100
0x200
0x300

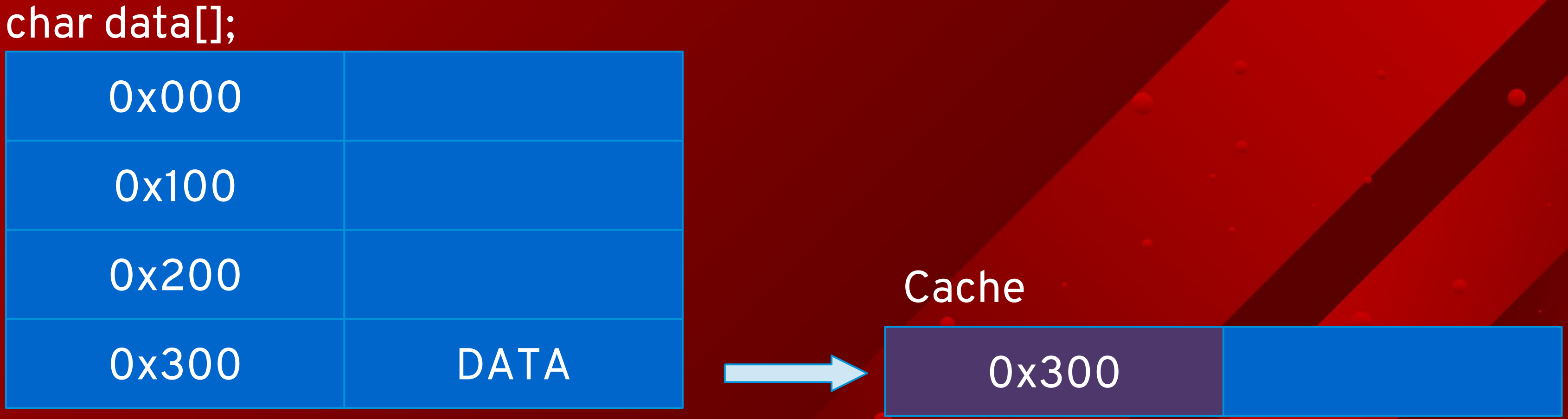
Meltdown

- Access to “data” element 0x100 pulls the corresponding entry into the cache



Meltdown

- Access to “data” element 0x300 pulls the corresponding entry into the cache



Meltdown: Speculative Execution

Entry	RegRename	Instruction	Deps	Ready?	Spec?
1	P1 = R1	R1 = LOAD SPEC_CONDITION	X	Y	N
2		TEST SPEC_CONDITION	1	Y	N
3		IF (SPEC_CONDITION) {	1	N	N
4	P2 = R2	R2 = LOAD KERNEL_ADDRESS	X	Y	Y*
5	P3 = R3	R3 = (((R2&1)*0x100)+0x200)	2	Y	Y*
6	P4 = R4	LOAD USER_BUF[R3]	3	Y	Y*

flags for future exception

Meltdown: Speculative Execution

Entry	RegRename	Instruction	Deps	Ready?	Spec?
1	P1 = R1	R1 = LOAD SPEC_CONDITION	X	Y	N
2		TEST SPEC_CONDITION	1	Y	N
3		IF (SPEC_CONDITION) {	1	N	N
4	P2 = R2	R2 = LOAD KERNEL_ADDRESS	X	Y	Y*
5	P3 = R3	R3 = (((R2&1)*0x100)+0x200)	2	Y	Y*
6	P4 = R4	LOAD USER_BUF[R3]	3	Y	Y*

should kill speculation here

Meltdown: Speculative Execution

Entry	RegRename	Instruction	Deps	Ready?	Spec?
1	P1 = R1	R1 = LOAD SPEC_CONDITION	X	Y	N
2		TEST SPEC_CONDITION	1	Y	N
3		IF (SPEC_CONDITION) {	1	N	N
4	P2 = R2	R2 = LOAD KERNEL_ADDRESS	X	Y	Y*
5	P3 = R3	R3 = (((R2&1)*0x100)+0x200)	2	Y	Y*
6	P4 = R4	LOAD USER_BUF[R3]	3	Y	Y*

really bad thing(™)

We mitigated “Meltdown” through “KPTI” (Kernel Page Table Isolation)*

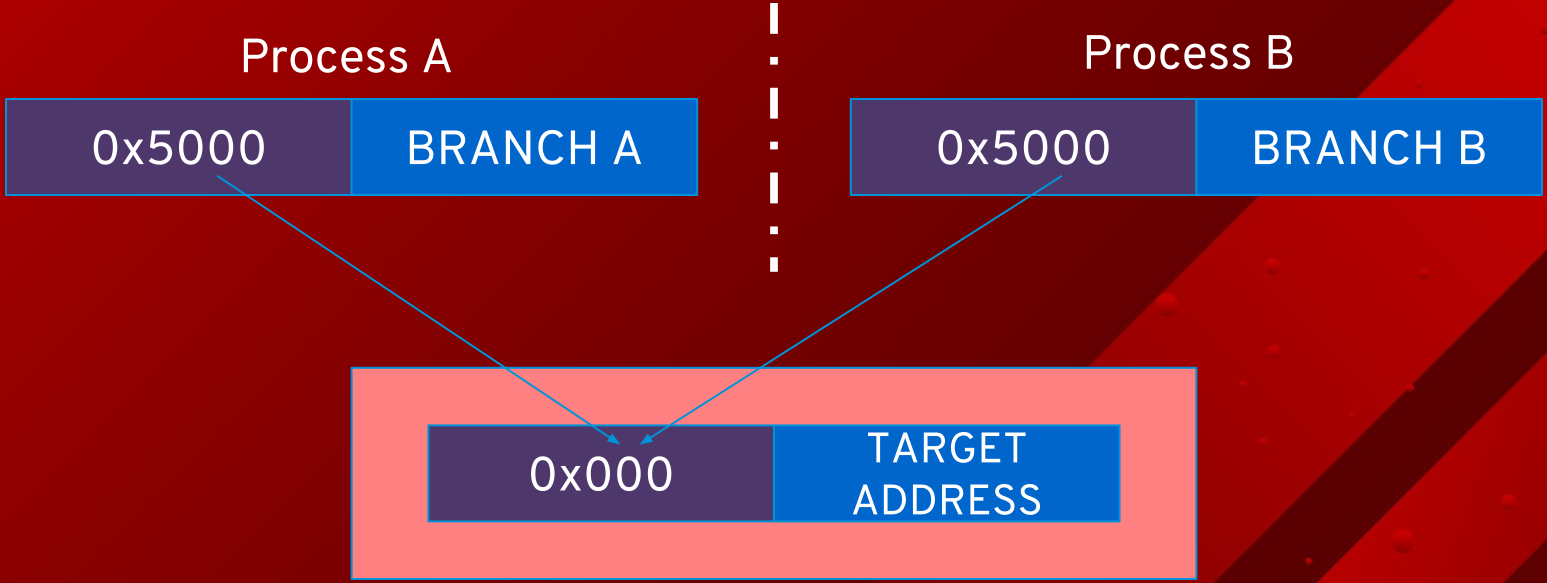
* Shipping in a currently supported RHEL kernel near you

“Meltdown” requires that we have valid kernel address translations (TLBs) so ensure we never do by spitting kernel and application page tables

Performance impact comes from the "trampoline" code that is needed to switch page tables on every kernel entry and exit back to a program

“Spectre” v2 abuses (indirect) branch predictors that don’t fully disambiguate between two different process “contexts”…

Spectre-v2 (branch predictor poisoning)



…we train the branch predictor in one context (program) such that it will guess a particular way in another context (the victim/kernel we want to control)

The attack relies upon finding “gadget” code already in the victim program

We mitigated “Spectre-v2” through a combination of firmware and OS level branch predictor control interfaces*

* Shipping in a currently supported RHEL kernel near you

CPUs have an ability to tweak certain behavior (e.g. “chicken bits”) or patch new operations into a small on-chip RAM via “microcode” updates

We use microcode for two things:

1. To restrict speculation across privilege boundaries (kernel entry)
2. To invalidate the branch predictor entries when switching programs

A later optimization ("retpolines") converts indirect function calls into fake return "trampolines". Avoids using the standard branch predictor hardware*

* Shipping default in a supported RHEL kernel near you, and you can even switch mitigations at run time

Another variant of Spectre (v1) requires minor code tweaks (barriers) that were added to the kernel source directly

Why these problems mattered

Nearly every more CPU affected (to differing extents)

A skilled attacker could have the ability to read any memory they desire

Some attacks cross guest/host boundaries

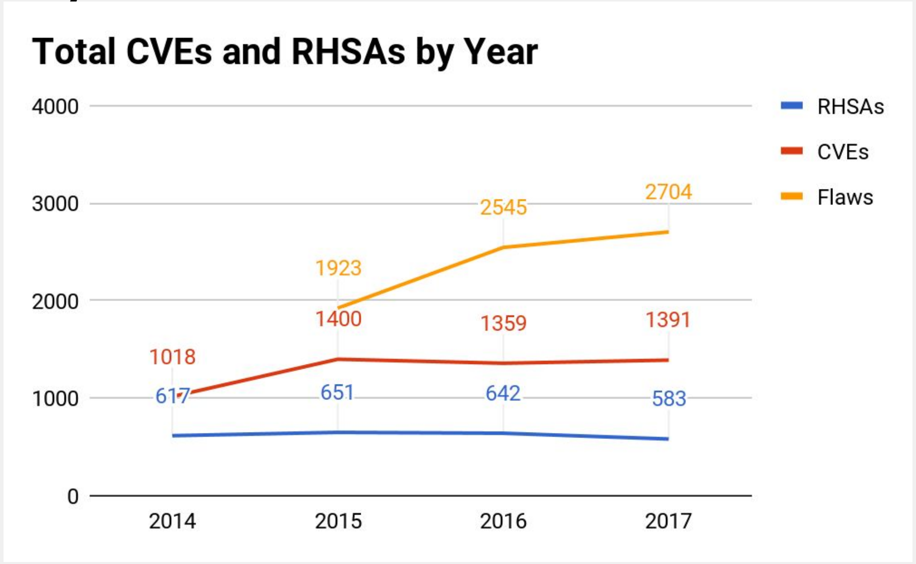
Virtually undetectable

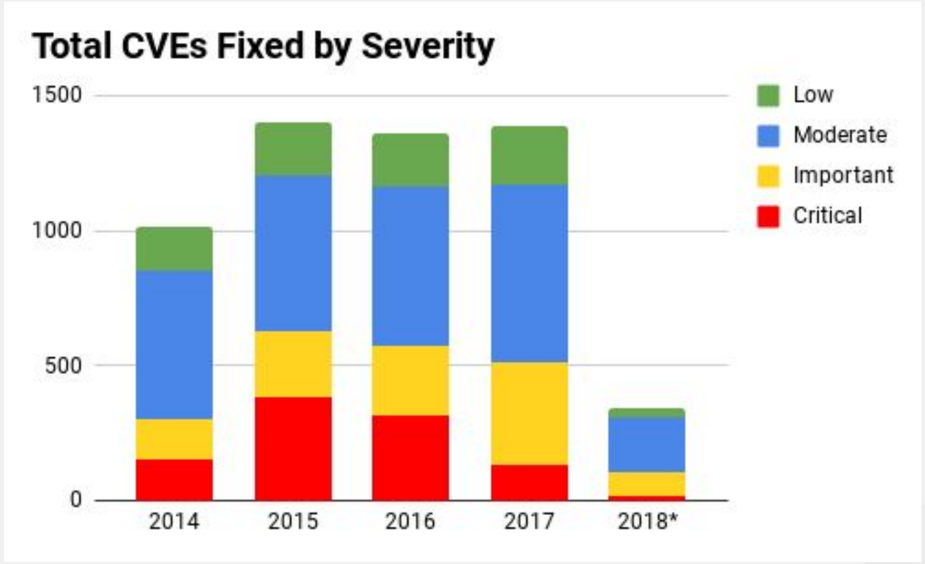
Why Red Hat was included

We have a very particular set of skills, skills we've acquired over our very long career

(the last 25 years and counting)

A snapshot of Red Hat Product Security over the years





https://www.redhat.com/security/data/metrics/
https://access.redhat.com/documentation/en-us/red_hat_security_data_api/0.1/html-single/red_hat_security_data_api/index

Coordinated Vulnerability Disclosure

Red Hat is part of a large group of vendor and community security teams

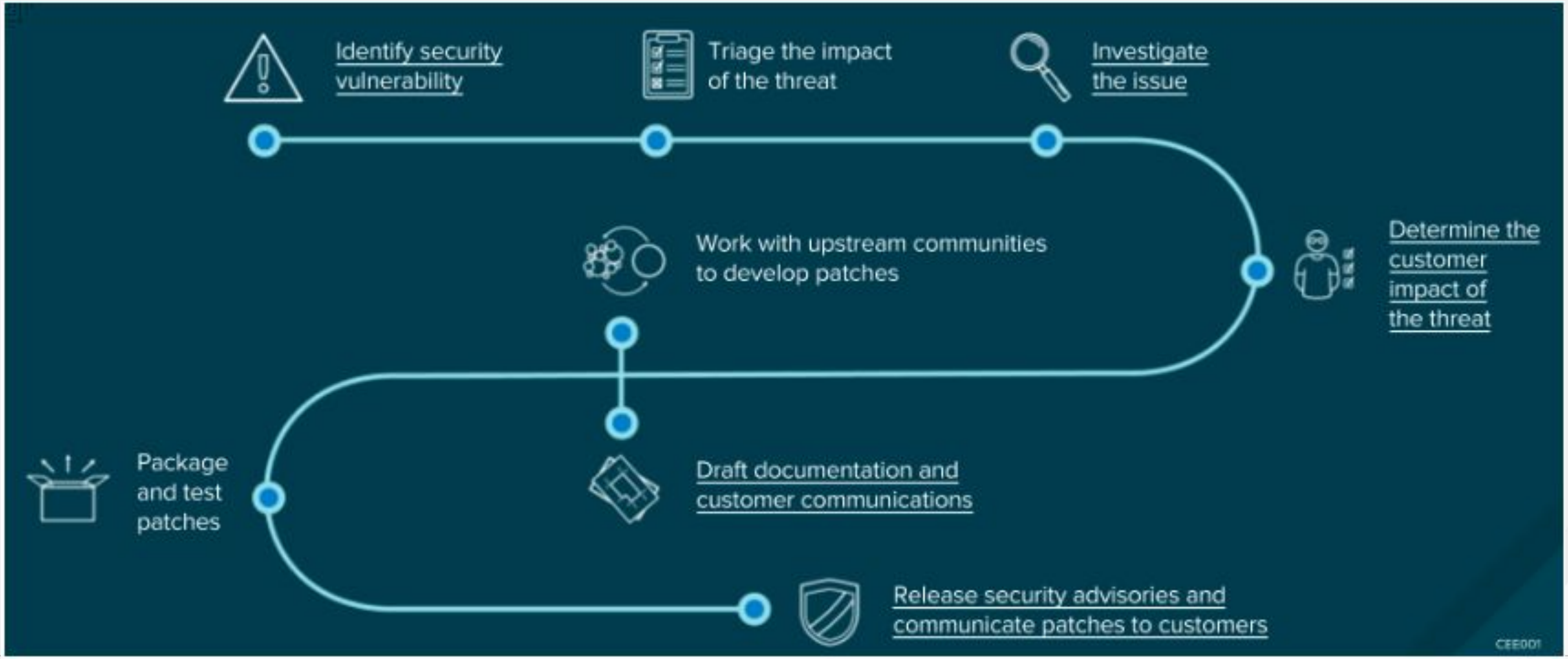
We use a process called Coordinated Vulnerability Disclosure(1) - The goal is to protect customers and the larger global computing community

Red Hat respects the wishes of the issue reporter on how they want the issue to be handled and how long to keep it secret

(1) https://resources.sei.cmu.edu/asset_files/SpecialReport/2017_003_001_503340.pdf

Red Hat’s Customer Security Awareness Program

Ride, ride my seesaw   Take this place   On this trip   Just for me


Identify security vulnerability
Triage the impact of the threat
Investigate the issue
Determine the customer impact of the threat
Work with upstream communities to develop patches
Draft documentation and customer communications
Package and test patches
Release security advisories and communicate patches to customers
CEE001

What we delivered to subscribers

15 kernels...3 times (4 if you count retpolines)

59 total patches (plus 15 more for retpolines) PLUS our complete OpenShift Online infrastructure (3 versions), plus all AMIs, ISOs, and base images

Over 60 engineers involved in developing, testing, and packaging the errata

Education video, 1 vulnerability article (translated into 6 languages), 4 kcs articles on specialized topics, 3 forms of customer emails, TAM T3, Performance webinar, and Portal Banners & Alerts

10,000+ engineering hours dedicate to resolving issues

Red Hat Product Security
CVE-2017-5753
CVE-2017-5715
CVE-2017-5754
3Jan2018
2017 CVE Response

To learn more....

Vulnerability article - https://access.redhat.com/security/vulnerabilities/speculativeexecution

3 minute Video - https://youtu.be/syAdX44pokE

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

CSAw process - https://access.redhat.com/articles/2968471

To sign up for RHSA announcements - https://www.redhat.com/wapps/ugc/protected/notif.html

RED HAT SUMMIT

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHat

youtube.com/user/RedHatVideos