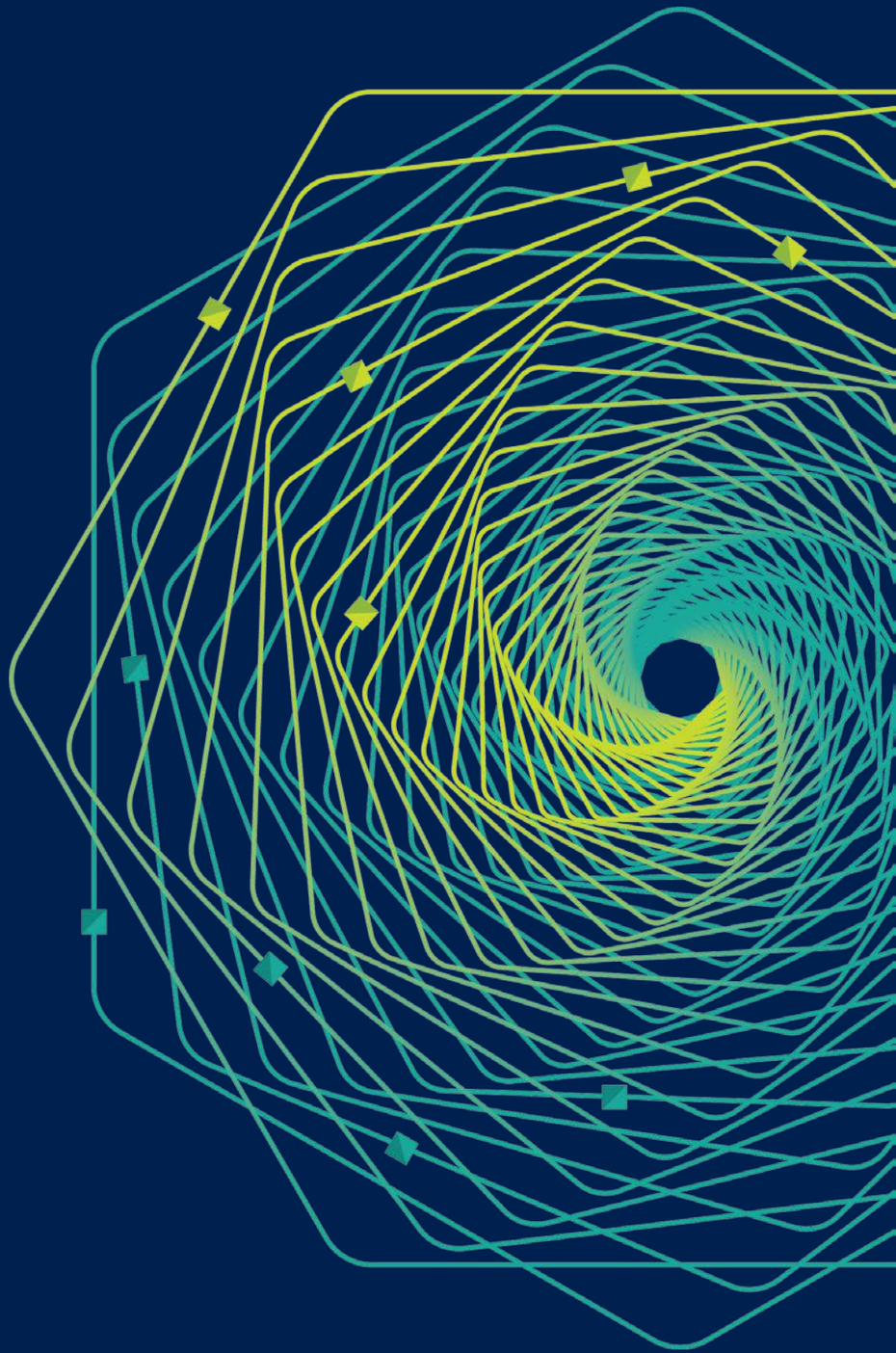
Microsoft

Research Faculty Summit 2018

Systems | Fueling future disruptions



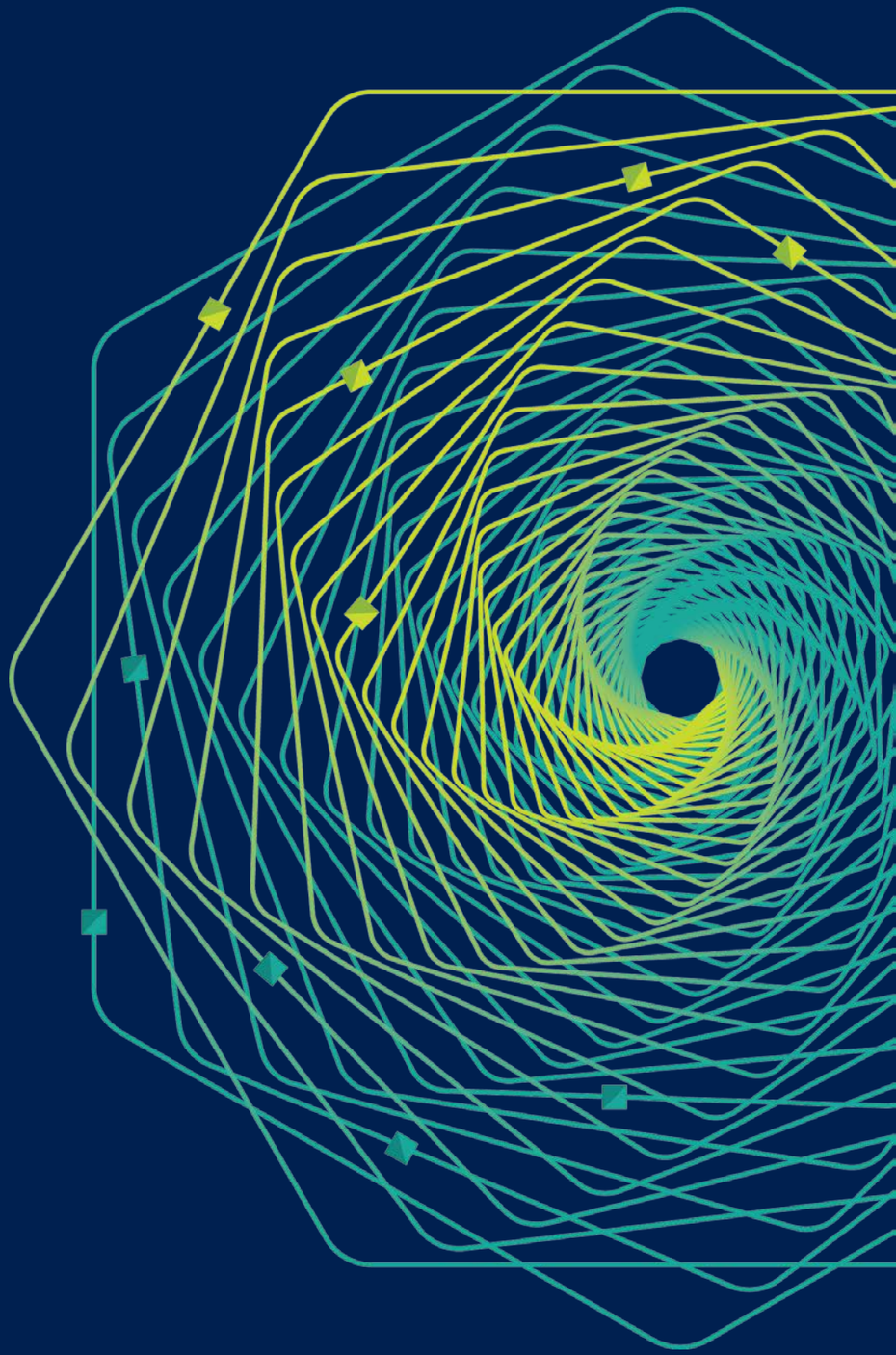
Microsoft

Wrangling the Ghost

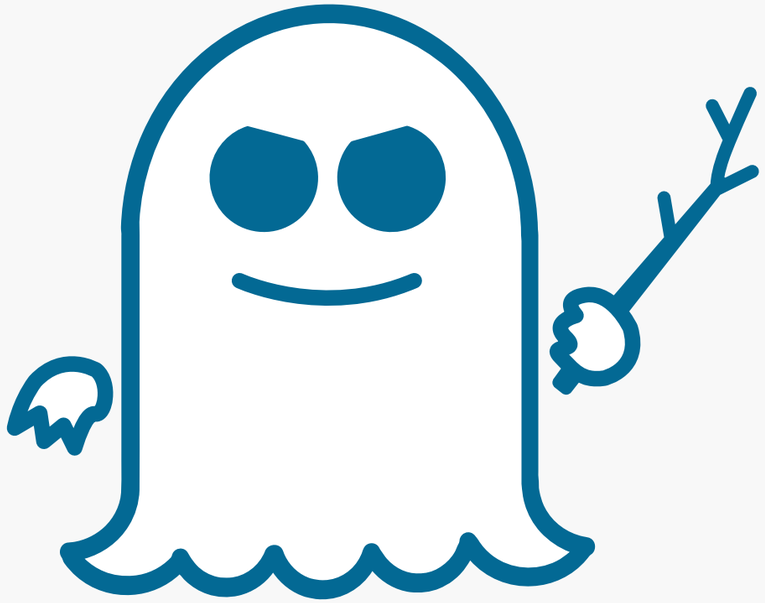
An Inside Story of Mitigating Speculative Execution Side Channel Vulnerabilities

Christopher Ertl

Security Engineer, Microsoft



You've most likely heard of Spectre & Meltdown



=

A new class of hardware vulnerability

Exploring a new vulnerability class

Microsoft first learned about these issues in June, 2017 when a CPU partner notified us

MSRC kicked off our "SSIRP" incident response process to drive remediation

- SSIRP drives cross-company and cross-industry response to critical security issues
- Eventually mobilized hundreds of people across Microsoft in response to this issue
- Disclosure date was eventually extended by 120 days due to complex nature of the mitigations required
- Advisory and security updates released January 3rd, 2018

Why does Microsoft care about these issues?

Because they are relevant to nearly every security boundary that software relies on

Virtualization-based isolation	Microsoft Azure, Hyper-V	Affected
Kernel-user separation	Windows	Affected
Process-based isolation	Windows	Affected
Language-based isolation	Microsoft Edge & Internet Explorer	Affected
Enclaves	Microsoft Azure, Windows	Affected

Impact: an attacker with local code execution can potentially read information that is stored in a higher privileged context

Systematization of Spectre and Meltdown

A taxonomy and framework for reasoning about speculative execution side channels

Parallelism and speculation

- We usually think of programs as a recipe
  - Instructions are sequentially executed one after the other
  - As it turns, out this sequential approach is pretty slow

- Modern high performance CPUs do many tasks at once

Pipeline	Instructions are put on an “assembly line” and different jobs are done in different stages
Superscalar	Multiple instructions are executed at once
Out-of-order execution	Instructions are executed as dependencies are resolved and resources are available
Speculative execution	Instructions are executed based on predictions

General definition of speculative execution

- Speculative execution: when the pipeline works on information that may not be correct if a program were executing like a recipe

- Speculative execution can consist of
  - Predicted conditional logic
  - Predicted instruction pointer (branch targets)
  - Predicted register values
  - Deferred error handling
  - And so on…

Spectre and Meltdown

- Fundamental idea of Spectre & Meltdown
  - Not everything is thrown away when speculative execution is unrolled
  - By carefully examining things like caches, results can be reestablished
  - These results may contain private data

Spectre (variant 1): conditional branches

A conditional branch can potentially mispredict, thus leading to a speculative out-of-bounds load that feeds a second load, thus creating cache side effects based on a secret. The attacker can train the branch to speculatively run the code.

`if (untrusted_index < length) {`	This can mispredict executing the below lines with any value of untrusted_index
`  char value = buf[untrusted_index];`	Loads nearly arbitrary memory
`  char value2 = buf2[value * 0x40];`	Loads the cache as an artifact of the value
`}`	

Consequence

If an attacker can find/create & execute this code in Hypervisor/Kernel/Enclave/sandbox, they can read the memory

Spectre (variant 2): indirect branches

An indirect branch can potentially mispredict the branch target, thus leading to speculative execution from an attacker controlled target address which could perform a load and feed that value to a second load

`0x4000: JMP RAX ; RAX = 0x5000` `....`	This can mispredict the target address, thus speculative executing anywhere

`0x6000:  MOVZX RCX, BYTE PTR [RCX]`
`SHL  RCX, 6`
`MOV  RCX, [RDX+RCX]`
	Loads any memory at RCX
	Multiply by 0x40 (cacheline size)
	Loads the cache as an artifact of the value

Consequence

If attacker can find/create & execute this code in Hypervisor/Kernel/Enclave/sandbox, they can read the memory

Meltdown (variant 3): exception deferral

Exception delivery may be deferred until instruction retirement, thus allowing data that should be inaccessible to be speculatively forwarded onto other instructions

```
TEST  RAX, RAX
JE    Skip
MOVZX RCX, BYTE PTR [KERNEL_ADDR]
SHL   RCX, 6
MOV   RCX, [Buf2+RCX]
```

This can mispredict the target address, thus speculative executing anywhere

Fetch any kernel address. Error/Roll back arrives delayed

Multiply by 0x40 to store information in the cache

Consequence

An unprivileged user mode process can read kernel memory

Why create a taxonomy?

- Designing robust mitigations requires a systematic approach

- Being systematic about a class of vulnerabillities requires a taxonomy

Building a taxonomy

4 steps are required of an attacker to successfully launch any speculative side channel attack

	Requirement	Taxonomy
Speculation	1. Gaining speculation	Speculation primitive
	2. Maintaining speculation	Windowing gadget
Side channel	3. Persisting the results	Disclosure gadget
	4. Observing the results	Disclosure primitive

If any of these 4 components are not present, there is no speculative side channel

Gaining speculation: speculation primitives

To have a speculative side channel, the CPU must be put in a situation where it will speculate

Spectre variant 1	Conditional branches are predicted on past behavior, thus we can train them
Spectre variant 2	Indirect branches can be trained in place like conditional branches, or since not all bits are used for prediction, they can be trained in an attacker controlled context
Meltdown	The CPU may defer exceptions and may speculatively forward data on to dependent instructions

Maintaining speculation: windowing gadgets

- An attacker can execute code speculatively
  - Starting with entering speculation
  - Ending with CPU detecting and rectifying mis-speculation

- To win this race condition, an attacker needs a windowing gadget
  - Allows for out-of-order execution
  - Can occur naturally in code
  - Can sometimes be engineered by an attacker
  - Window size is determined by hardware, dependencies and resource congestion

Side channel basics

Side channels typically contain 3 phases of which 2 are strictly required

Priming	Getting the system into a known initial state (e.g. flushing cache lines)
Triggering	Actively or passively causing the victim to execute
Observing	Observe if state is changed and thereby infer information from this

Persisting results: disclosure gadgets

- When speculation is rolled back information is lost unless exfiltrated by side channel

- Thus, an attacker needs to write to a side channel within the speculative window
  - Example: speculative execution changes the cache state

Observing the results: disclosure primitives

- Finally the attacker needs to read the results from the side channel
  - Example: check if a cache line was loaded

The four components of speculation techniques

Speculation primitive	Example

Conditional branch misprediction	`if (n < *p) {`
	`  // can speculate when n >= *p`
	`}`

Indirect branch misprediction	`// can speculate wrong branch target` `(*FuncPtr)();`

Exception delivery	`// may do permission check at`
	`// retirement`
	`value = *p;`

Disclosure primitive	Example
FLUSH+RELOAD	Priming phase: flush candidate cache lines Trigger phase: cache line is loaded based off secret Observing phase: load candidate cache lines, fastest access may be signal
EVICT+TIME	Priming phase: evict congruent cache line Trigger phase: cache line is loaded based off secret Observing phase: measure time of operation, slowest operation may be signal
PRIME+PROBE	Priming phase: load candidate cache lines Triggering phase: cache set is evicted based off secret Observing phase: load candidate cache lines, slowest access may be signal

Windowing gadget	Example
Non-cached load	`// *p not present in cache` `value = *p;`
Dependency chain of loads	`value = ********p;`
Dependency chain of ALU operations	`value += 10;` `value += 10;` `value += 10;`
…	

Disclosure gadget	Example
One level of memory indirection, out-of-bounds	`if (x < y)` `  return buf[x];`

Two levels of memory indirection, out-of-bounds	`if (x < y) {`
	`  n = buf[x];`
	`  return buf2[n];`
	`}`

Three levels of memory indirection, out-of-bounds	`if (x < y) {`
	`  char *p = buf[n];`
	`  char b = *p;`
	`  return buf2[b];`
	`}`

…	

Relevance to software security models

Attack category	Attack scenario	Conditional branch misprediction	Indirect branch misprediction	Exception delivery
Inter-VM	Hypervisor-to-guest			
	Host-to-guest			
	Guest-to-guest			
Intra-OS	Kernel-to-user			
	Process-to-process			
	Intra-process			
Enclave	Enclave-to-any			

Legend: Applicable | Not applicable

Mitigating speculative execution side channel vulnerabilities

Using our taxonomy to help mitigate Spectre, Meltdown, and speculative execution side channels as a whole

Defining our mitigation tactics

The systematization we developed provides the basis for defining our mitigation tactics

Prevent speculation techniques	Prevent a speculation primitive from executing a disclosure gadget
Remove sensitive content from memory	Ensure there is no sensitive information in memory that could be read by a speculation technique
Remove observation channels	Remove channels for communicating information via speculation techniques

No silver bullet; a combination of software, hardware, and scenario-specific mitigations

Preventing speculation techniques

Goal: prevent a speculation primitive from executing a disclosure gadget

Speculation barrier via execution serializing instruction

Speculative execution can be prevented through the use of a serializing instruction

Explicit serialization

```
if (untrusted_index < length) {
    _mm_lfence(); // barrier for speculation
    char value = buf[untrusted_index];
    char value2 = buf2[value * 0x40];
}
```

Architectural instruction that acts as a speculation barrier

LFENCE on AMD/Intel and CSDB on ARM

Implicit serialization

```
if (untrusted_index < length) {
    // cmp untrusted_index, length
    // xor reg,reg
    // cmovae untrusted_index, reg
    char value = buf[untrusted_index];
    char value2 = buf2[value * 0x40];
}
```

Force safe behavior in the speculative path by bounding array indices

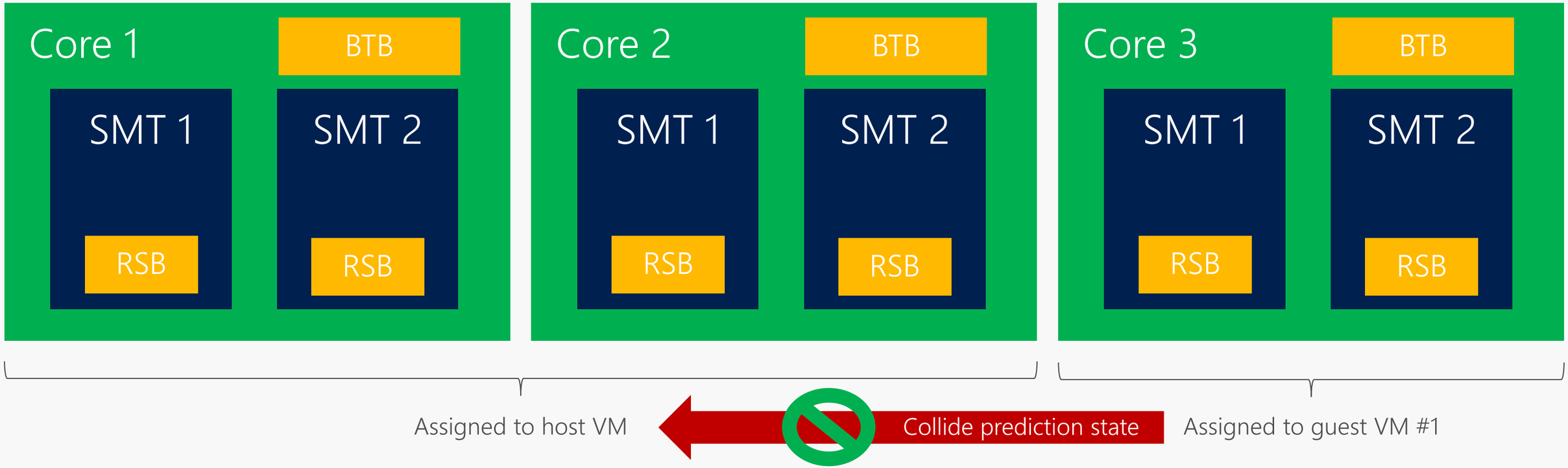
CMOV-based implicit serialization is safe on existing CPUs

- Microsoft Visual C++ compiler supports /Qspectre which has narrow heuristics to find and instrument variant 1
- Microsoft Edge and Internet Explorer JavaScript engines have code generation mitigations for variant 1

Security domain CPU core isolation

CPUs typically store prediction state in per-core or per-SMT caches

Isolating workloads to distinct cores can prevent colliding of prediction state



- Microsoft Hyper-V supports minimum root (“minroot”) and CPU groups which can isolate VMs to cores

Indirect branch speculation barrier on demand & mode change

Cross-mode attacks on indirect branch misprediction can be mitigated with new CPU features

Intel, AMD, and ARM have created or defined interfaces to manage indirect branch predictions

Indirect Branch Restricted Speculation (IBRS)	Indirect Branch Prediction Barrier (IBPB)	Single-Thread Indirect Prediction Barrier (STIBP)
When IBRS=1, less-privileged modes cannot influence indirect branch predictions of higher-privileged modes Kernel and/or hypervisor can set IBRS=1 on entry to prevent less privileged modes from attacking them	When IBPB=1, indirect branch prediction state is flushed (BTB and RSB) Kernel and/or hypervisor can write this when switching process or VM contexts to prevent cross-process and cross-VM attacks	When STIBP=1, sibling SMTs cannot influence one another's indirect branch predictions Processes can request that the kernel set this to prevent cross-process SMT-based attacks on indirect branch prediction

- All supported versions of Windows support Indirect Branch Control (IBC) features from Intel and AMD
- IBC is enabled by default on Windows Client and is disabled by default on Windows Server
- Intel and AMD have released microcode updates

Non-speculated or safely-speculated indirect branches

Some indirect branches are not predicted or can be safely predicted		
FAR JMP and FAR RET are not predicted on Intel CPUs	**RDTSCP or LFENCE before indirect JMP is safe on AMD CPUs**	**Indirect calls and jumps can be transformed into "retpolines"**
Indirect calls and jumps can be transformed into FAR JMP on Intel CPUs	Indirect calls and jumps can be transformed into RDTSCP or LFENCE before indirect JMP on AMD CPUs	Google proposed "retpoline" which transforms indirect calls and jumps into "retpoline" stubs

- Hyper-V hypervisor transforms all indirect calls to FAR JMP on Intel and RDTSCP-before-JMP on AMD
- Windows kernel is exploring a hybrid retpoline + IBC model as a possible way to help improve performance
- These solutions require rebuilding the world which limits viable use cases

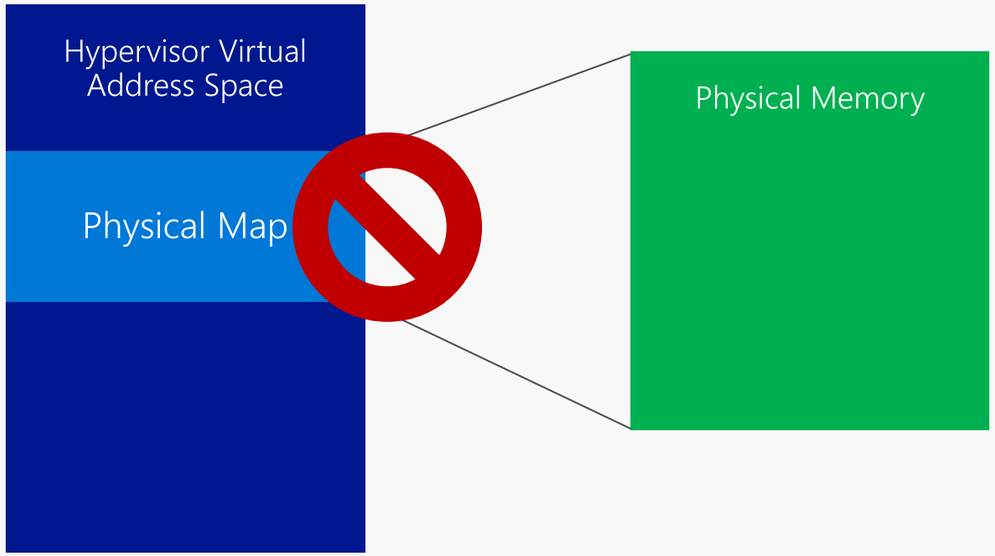
Removing sensitive content from memory

Goal: ensure there is no sensitive information in memory that could be read by a speculation technique

Hypervisor address space segregation

Hyper-V's hypervisor historically mapped all physical memory into HV address space

Removing the physical map helps eliminate cross-VM secrets that may be subject to disclosure



- Hyper-V hypervisor now maps guest physical memory on-demand, limiting physical memory that is mapped

Split user and kernel page tables (KVA Shadow)

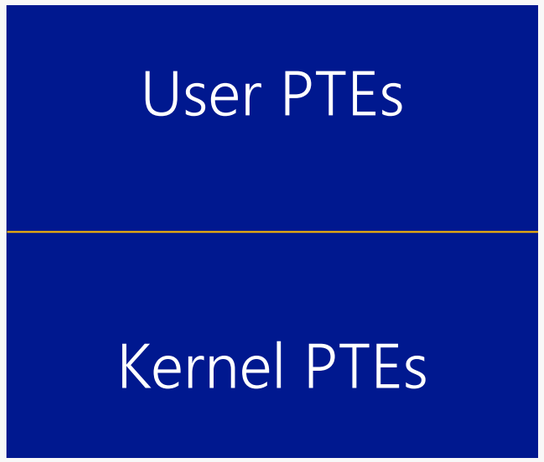
Variant 3 was exploitable because kernel memory was part of the address space even in user mode

KVA Shadow creates split kernel/user page tables which makes kernel memory inaccessible in user mode

Without KVA Shadow

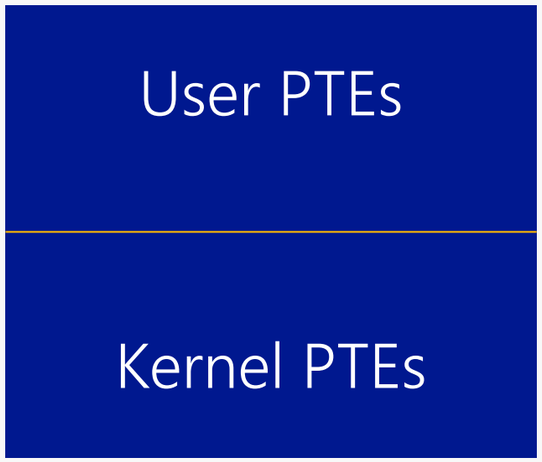
User mode and kernel mode share the same page tables

User page directory base



=

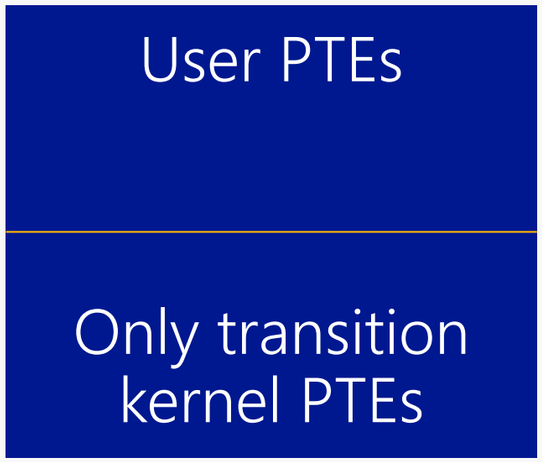
Kernel page directory base



With KVA Shadow

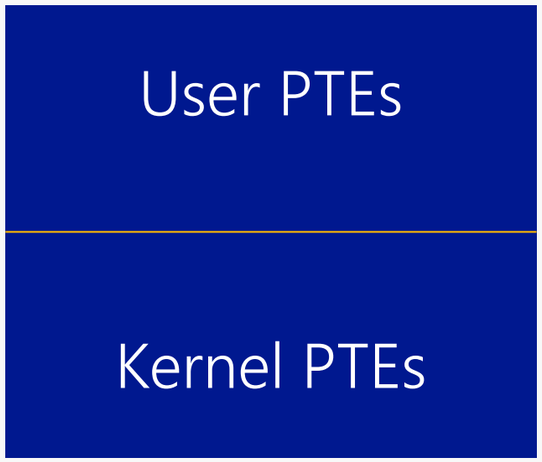
User mode and kernel mode have their own page directory base
CR3 swaps between them on kernel entry and exit

User page directory base



≠

Kernel page directory base



- All supported versions of Windows support KVA Shadow
- KVA Shadow is enabled by default on Windows Client and is disabled by default on Windows Server

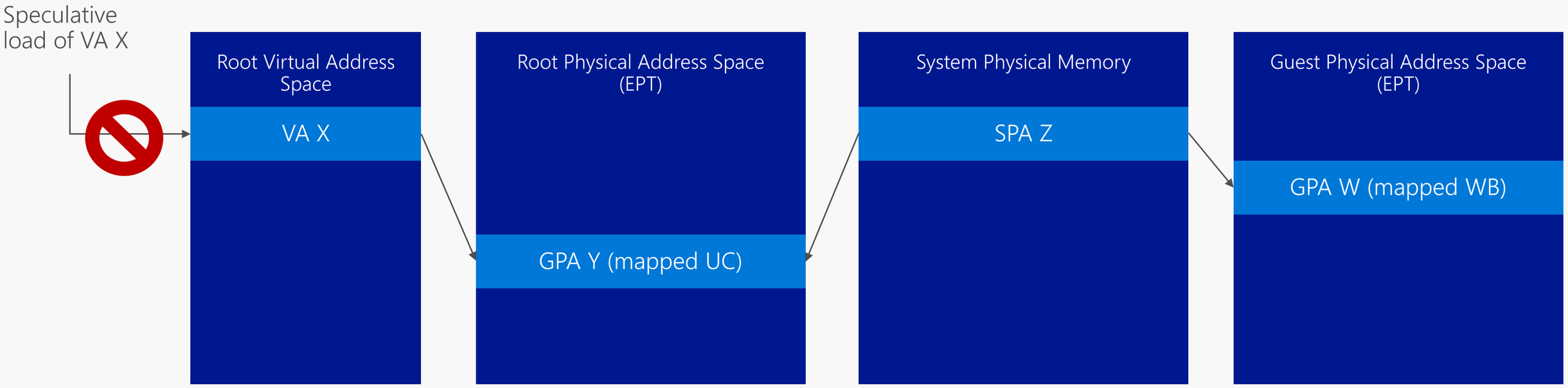
Removing observation channels

Goal: remove channels for communicating information via speculation techniques

Map guest memory as UC in root EPT

FLUSH+RELOAD relies on shared cache lines for host-to-guest disclosure

Hypervisors can map guest physical memory as UC into the root partition's extended page tables (EPT)

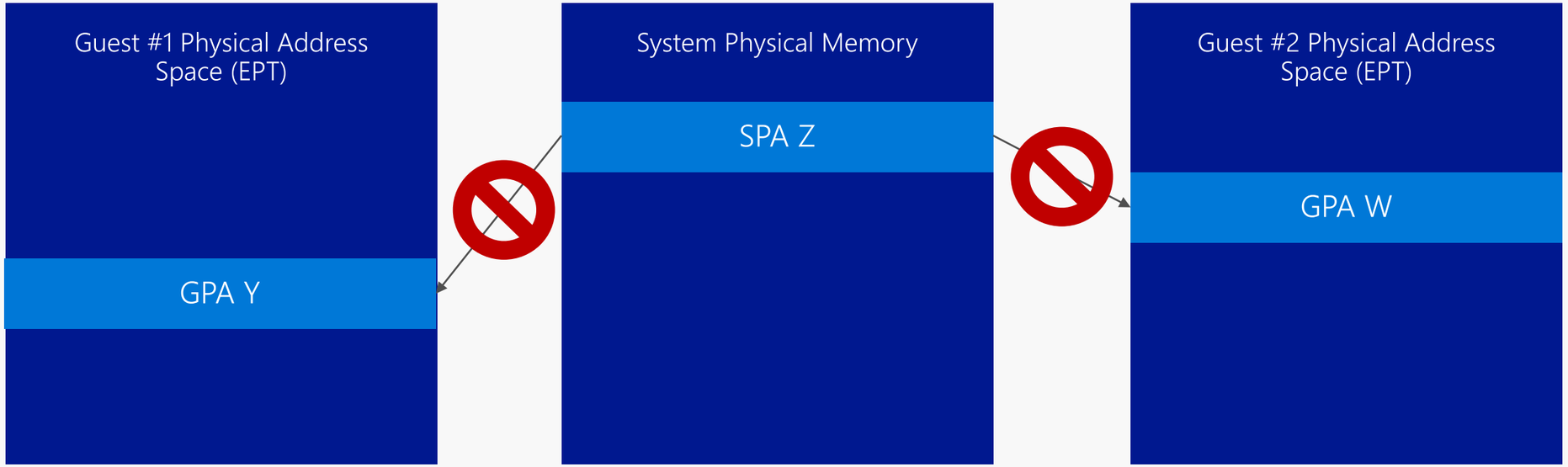


UC mapping into root prevents speculative load of a shared cache line, generically mitigating host-to-guest FLUSH+RELOAD

Do not share physical pages across guests

FLUSH+RELOAD relies on shared cache lines for guest-to-guest attacks

Hypervisor can ensure that physical memory is not shared between guests

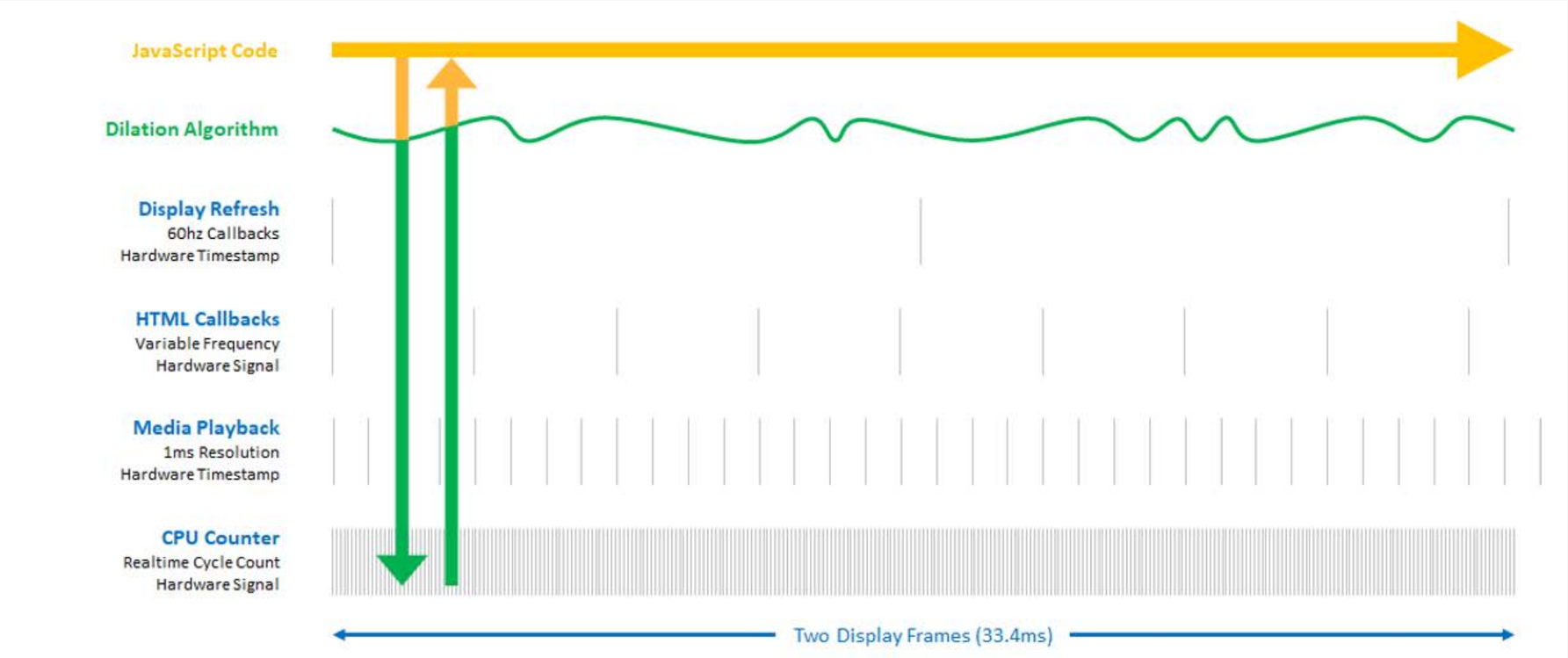


Absence of shared physical pages between guests provides a mitigation for guest-to-guest FLUSH+RELOAD

Decrease browser timer precision

Speculative execution side channels rely on precise timing for high bandwidth signal detection

Browsers can reduce the precision of timers that are visible to JavaScript that attackers specify



- Microsoft Edge and Internet Explorer both decrease timer precision and add random jitter

Closing remarks

Mitigation relationship to attack scenarios & primitives

Mitigation Tactic	Mitigation Name	Attack category			Speculation primitive		
		Inter-VM	Intra-OS	Enclave	Conditional branch misprediction	Indirect branch misprediction	Exception delivery
Prevent speculation techniques	Speculation barrier via execution serializing instruction	■	■	■	■		
	Security domain CPU core isolation	■			■	■	
	Indirect branch speculation barrier on demand and mode change	■	■	■		■	
	Non-speculated or safely-speculated indirect branches	■	■	■		■	
Remove sensitive content from memory	Hypervisor address space segregation	■			■	■	
	Split user and kernel page tables ("KVA Shadow")		■				■
Remove observation channels	Map guest memory as noncacheable in root extended page tables	■			■	■	
	Do not share physical pages across guests	■			■	■	
	Decrease browser timer precision		■		■	■	■

Legend:

Applicable	Not applicable

How should developers think about each variant?

Variant	Conceptualization
Variant 1 (CVE-2017-5753)	This is a hardware vulnerability class that requires software changes in order to mitigate. No universal mitigation for this variant exists today.
Variant 2 (CVE-2017-5715)	This is a hardware vulnerability that can be mitigated through a combination of OS and firmware changes.
Variant 3 (CVE-2017-5754)	This is a hardware vulnerability that can be mitigated through OS changes to create split user/kernel page tables.

New variants & mitigations

Since January, research interest has increased & new variants have been identified

Disclosed	Variant	Speculation primitive category	Mitigation
May, 2018	Speculative Store Bypass (CVE-2018-3639)	Memory access misprediction (new category)	• Disable speculative store bypass optimization • Speculation barrier prior to unsafe store
June, 2018	Lazy FP State Restore (CVE-2018-3665)	Exception delivery (same as Meltdown)	• Use eager restore of FP state (rather than lazy restore)
July, 2018	Bounds Check Bypass Store	Conditional branch misprediction (same as Spectre variant 1)	• Speculation barrier as required
July, 2018	NetSpectre	Conditional branch misprediction (same as	• Speculation barrier as required

We expect speculative execution side channels to be an ongoing subject of research

Resources

- Microsoft Speculative Execution Side Channel Bounty
  - https://aka.ms/sescbounty
- C++ developer guidance for speculative execution side channels
  - https://aka.ms/sescdevguide
- Technical analysis
  - https://blogs.technet.microsoft.com/srd/2018/03/15/mitigating-speculative-execution-side-channel-hardware-vulnerabilities/
  - https://blogs.technet.microsoft.com/srd/2018/03/23/kva-shadow-mitigating-meltdown-on-windows/
  - https://blogs.technet.microsoft.com/srd/2018/05/21/analysis-and-mitigation-of-speculative-store-bypass-cve-2018-3639/

Thank you!

A huge THANK YOU to:

- The security researchers who identified these issues
- Everyone in the industry who worked collaboratively to help protect the world
- The many individuals & teams at Microsoft who helped mitigate these issues ☺

Acknowledgements

Jann Horn of Google Project Zero
Paul Kocher
Moritz Lipp from Graz University of Technology
Daniel Genkin from University of Pennsylvania and University of Maryland
Daniel Gruss from Graz University of Technology
Werner Haas of Cyberus Technology GmbH
Mike Hamburg of Rambus Security Division
Stefan Mangard from Graz University of Technology
Thomas Prescher of Cyberus Technology GmbH
Michael Schwarz from Graz University of Technology
Yuval Yarom of The University of Adelaide and Data61
Additional information on the Meltdown and Spectre attacks can be found at their respective web sites.
Anders Fogh of GDATA Advanced Analytics

Microsoft