KASLR is Dead: Long Live KASLR

D. Gruss, M. Lipp, M. Schwarz, R. Fellner, C. Maurice, S. Mangard

July 5, 2017—ESSoS'17

Graz University of Technology

Motivation

- KASLR to make memory corruption attacks inpractical

Motivation

- KASLR to make memory corruption attacks inpractical
- Past year side-channel attacks have been published defeating KASLR

Motivation

- KASLR to make memory corruption attacks inpractical
- Past year side-channel attacks have been published defeating KASLR
- Is there a way to prevent these side-channel attacks?

Motivation

- KASLR to make memory corruption attacks inpractical
- Past year side-channel attacks have been published defeating KASLR
- Is there a way to prevent these side-channel attacks?
  - In Software?

Background

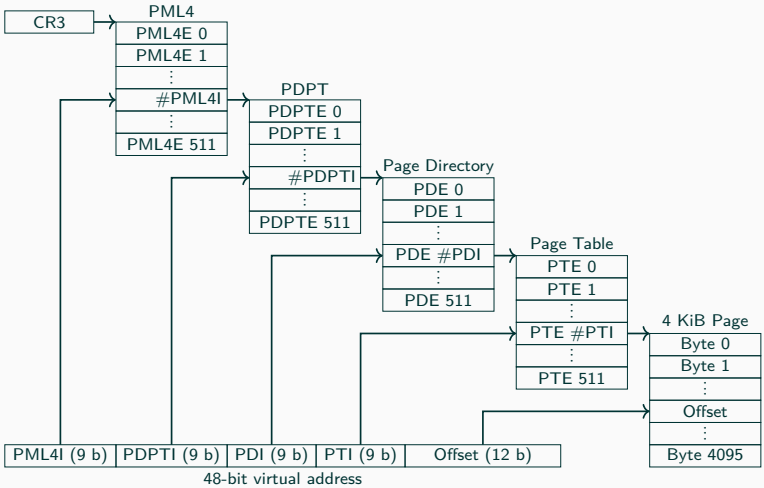
Virtual Memory

- Maps virtual addresses used by a program to physical addresses in DRAM

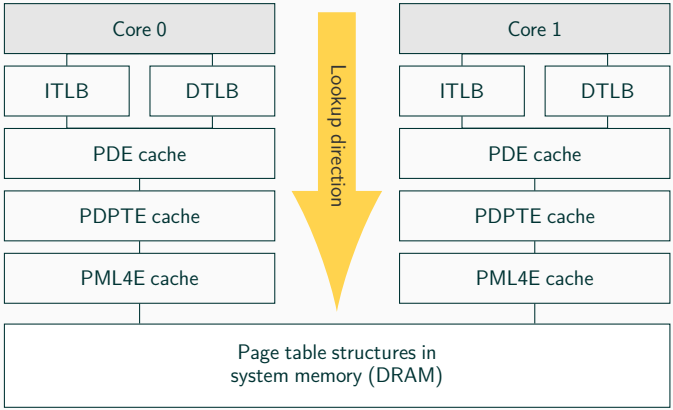
- Maps virtual addresses used by a program to physical addresses in DRAM
- Virtual address space is organized in pages

- Maps virtual addresses used by a program to physical addresses in DRAM
- Virtual address space is organized in pages
- Page tables are used to translate virtual to physical addresses

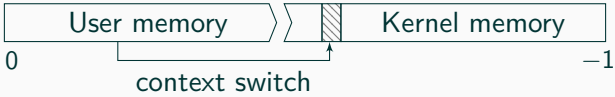
Address Translation on x86-64



Address Translation Caches



Address Space



- Many exploits rely on the knowledge of the memory location of a certain function

KASLR: Kernel Address Space Layout Randomization

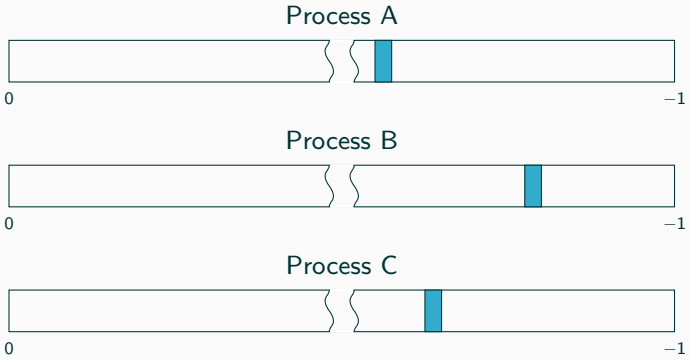
- Many exploits rely on the knowledge of the memory location of a certain function
- Statistical mitigation of memory corruption vulnerabilities

KASLR: Kernel Address Space Layout Randomization

- Many exploits rely on the knowledge of the memory location of a certain function
- Statistical mitigation of memory corruption vulnerabilities
- Randomizing core kernel image and device drivers position at boot time

KASLR: Kernel Address Space Layout Randomization

- Many exploits rely on the knowledge of the memory location of a certain function
- Statistical mitigation of memory corruption vulnerabilities
- Randomizing core kernel image and device drivers position at boot time
- Enabled in Linux 4.12 by default (May 2017)



- Driver is loaded to a different offset on every boot

Attacks against KASLR

TSX Attack

- 2016 by Jang et al. [Jan+16]
- Transactional Synchronization Extension (TSX)
  - Transaction aborts if conflict occurs

TSX Attack

- 2016 by Jang et al. [Jan+16]
- Transactional Synchronization Extension (TSX)
  - Transaction aborts if conflict occurs
  - Measure execution time in TSX abort handler

TSX Attack

- 2016 by Jang et al. [Jan+16]
- Transactional Synchronization Extension (TSX)
  - Transaction aborts if conflict occurs
  - Measure execution time in TSX abort handler
    - Fast: Address is mapped
    - Slow: Address is not mapped

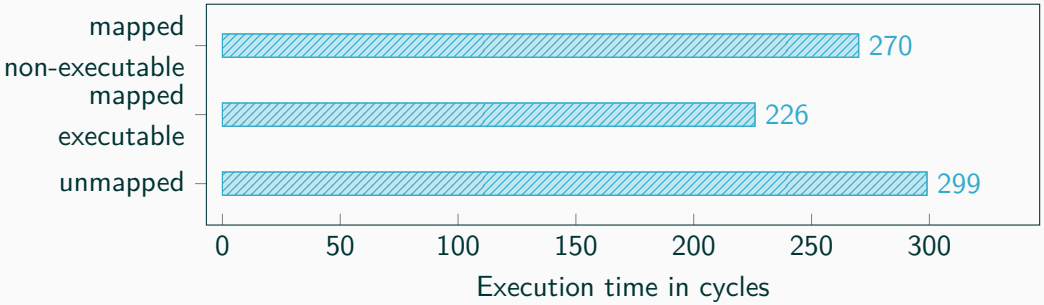
TSX Attack

- 2016 by Jang et al. [Jan+16]
- Transactional Synchronization Extension (TSX)
  - Transaction aborts if conflict occurs
  - Measure execution time in TSX abort handler
    - Fast: Address is mapped
    - Slow: Address is not mapped
- Reveal mapping status of each page

TSX Attack

- 2016 by Jang et al. [Jan+16]
- Transactional Synchronization Extension (TSX)
  - Transaction aborts if conflict occurs
  - Measure execution time in TSX abort handler
    - Fast: Address is mapped
    - Slow: Address is not mapped
- Reveal mapping status of each page
- Detect kernel modules with unique size signature

- 2016 by Jang et al. [Jan+16]
- Transactional Synchronization Extension (TSX)
  - Transaction aborts if conflict occurs
  - Measure execution time in TSX abort handler
    - Fast: Address is mapped
    - Slow: Address is not mapped
- Reveal mapping status of each page
- Detect kernel modules with unique size signature
- Less noisy than Double Page Fault Attack by Hund et al. [Hun+13]


mapped
non-executable
mapped
executable
unmapped
270
226
299
0
50
100
150
200
250
300
Execution time in cycles

Prefetch Side-Channel Attack

- 2016 by Gruss et al. [Gru+16]

Prefetch Side-Channel Attack

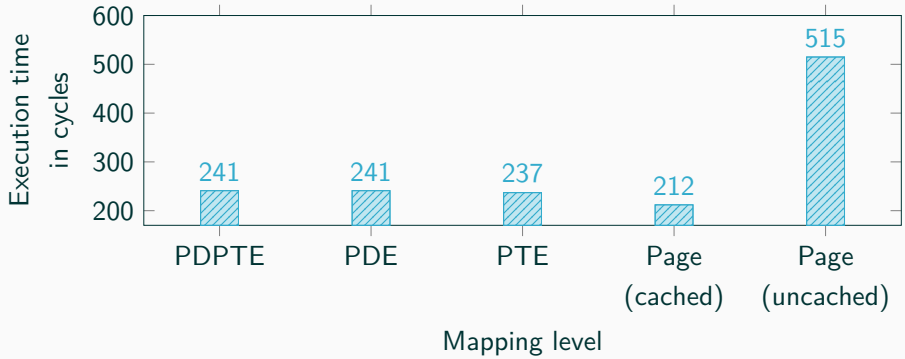
- 2016 by Gruss et al. [Gru+16]
- Execution time of Prefetch instruction varies
  - Depends on which address translation cache holds the right entry

Prefetch Side-Channel Attack

- 2016 by Gruss et al. [Gru+16]
- Execution time of Prefetch instruction varies
  - Depends on which address translation cache holds the right entry
- Reveal mapping status

Prefetch Side-Channel Attack

- 2016 by Gruss et al. [Gru+16]
- Execution time of Prefetch instruction varies
  - Depends on which address translation cache holds the right entry
- Reveal mapping status
- Allows to obtain physical address of virtual address


Execution time in cycles
600
500
400
300
200
241
241
237
212
515
PDPTE
PDE
PTE
Page (cached)
Page (uncached)
Mapping level

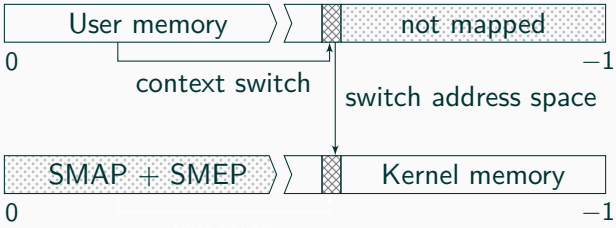
KASLR is Dead: Long Live KASLR

Idea

- KAISER: Kernel Address Isolation to have Side channels Efficiently Removed

Idea

- KAISER: Kernel Address Isolation to have Side channels Efficiently Removed
- Kernel Address Isolation: Separate kernel space and user space



- **Challenge 1:** Threads cannot use the same page table structures in user space and kernel space without a huge synchronization overhead.

- **Challenge 1:** Threads cannot use the same page table structures in user space and kernel space without a huge synchronization overhead.
- **Challenge 2:** Several locations must be valid for both user space and kernel space during context switches. Identify them.

- **Challenge 1:** Threads cannot use the same page table structures in user space and kernel space without a huge synchronization overhead.
- **Challenge 2:** Several locations must be valid for both user space and kernel space during context switches. Identify them.
- **Challenge 3:** Switching the address space incurs an implicit TLB flush. Performance impact.

Solving Challenge 1

- Every process has two address spaces:

Solving Challenge 1

- Every process has two address spaces:
  - Kernel Address Space: Kernel mapped, user space mapped and protected with SMAP and SMEP

Solving Challenge 1

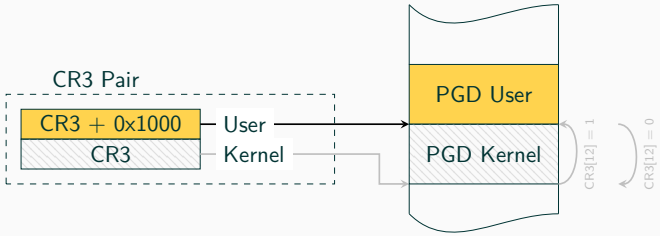
- Every process has two address spaces:
  - Kernel Address Space: Kernel mapped, user space mapped and protected with SMAP and SMEP
  - Shadow Address Space: Userspace mapped, Kernel not mapped

Solving Challenge 1

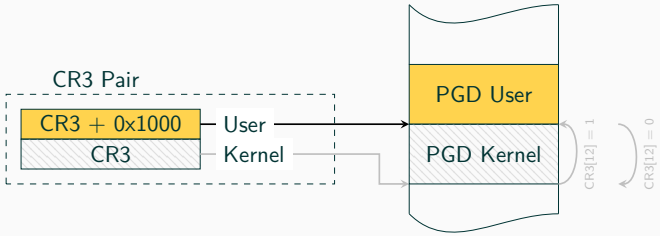
- Every process has two address spaces:
  - Kernel Address Space: Kernel mapped, user space mapped and protected with SMAP and SMEP
  - Shadow Address Space: Userspace mapped, Kernel not mapped
- Switching between the address space:

Solving Challenge 1

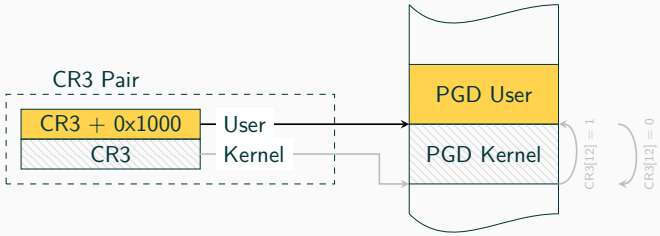
- Every process has two address spaces:
  - Kernel Address Space: Kernel mapped, user space mapped and protected with SMAP and SMEP
  - Shadow Address Space: Userspace mapped, Kernel not mapped
- Switching between the address space:
  - Update CR3 with corresponding PML4



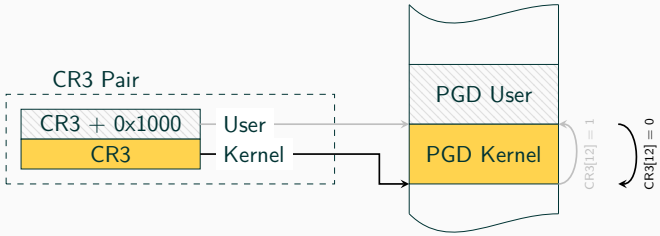
- Power-of-two offset between kernel and shadow PML4



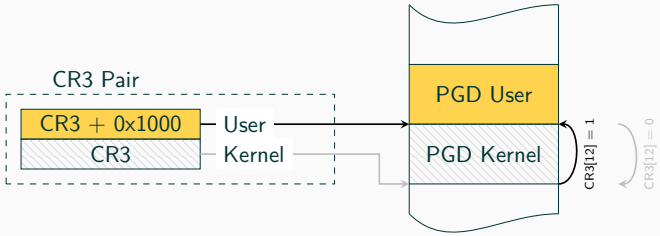
- Power-of-two offset between kernel and shadow PML4
- 8kb-aligned physical memory block to store both PML4s



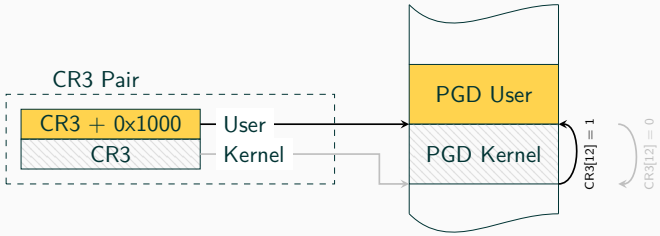
- Power-of-two offset between kernel and shadow PML4
- 8kb-aligned physical memory block to store both PML4s
- Toggle bit 12 of the physical address to switch between mappings



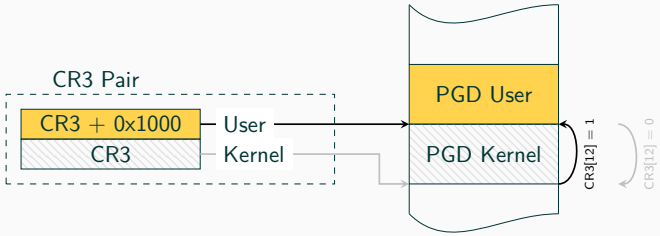
- Power-of-two offset between kernel and shadow PML4
- 8kb-aligned physical memory block to store both PML4s
- Toggle bit 12 of the physical address to switch between mappings



- Power-of-two offset between kernel and shadow PML4
- 8kb-aligned physical memory block to store both PML4s
- Toggle bit 12 of the physical address to switch between mappings



- Power-of-two offset between kernel and shadow PML4
- 8kb-aligned physical memory block to store both PML4s
- Toggle bit 12 of the physical address to switch between mappings
- No memory lookups



- Power-of-two offset between kernel and shadow PML4
- 8kb-aligned physical memory block to store both PML4s
- Toggle bit 12 of the physical address to switch between mappings
- No memory lookups
- Only a single scratch register

- Previous work suggested that only a portion of the interrupt dispatcher needs to be mapped

- Previous work suggested that only a portion of the interrupt dispatcher needs to be mapped
- Not practical
  - In reality, much more has to be mapped

Solving Challenge 2

- Interrupt Descriptor Table (IDT)

Solving Challenge 2

- Interrupt Descriptor Table (IDT)
- Interrupt entry and exit .text

- Interrupt Descriptor Table (IDT)
- Interrupt entry and exit .text
- Multi-threaded applications running on different cores:
  - per-CPU memory regions
  - interrupt request (IRQ) stack and vector
  - global descriptor table (GDT)
  - task state segment (TSS)
  - thread stacks

Solving Challenge 2

- Design of kernel is based upon the capability to access user space addresses from kernel mode

Solving Challenge 2

- Design of kernel is based upon the capability to access user space addresses from kernel mode
- SMEP: Protects against executing user space code from kernel mode

Solving Challenge 2

- Design of kernel is based upon the capability to access user space addresses from kernel mode
- SMEP: Protects against executing user space code from kernel mode
- SMAP: Invalid user memory references in kernel mode

- Minimize the number of implicit TLB flushes

Solving Challenge 3

- Minimize the number of implicit TLB flushes
- x86 performs implicit TLB flush on every CR3 update

Solving Challenge 3

- Minimize the number of implicit TLB flushes
- x86 performs implicit TLB flush on every CR3 update
- PTE global bit
  - Preserve mappings that exist in every process
  - Excluded from implicit TLB flushes

Solving Challenge 3

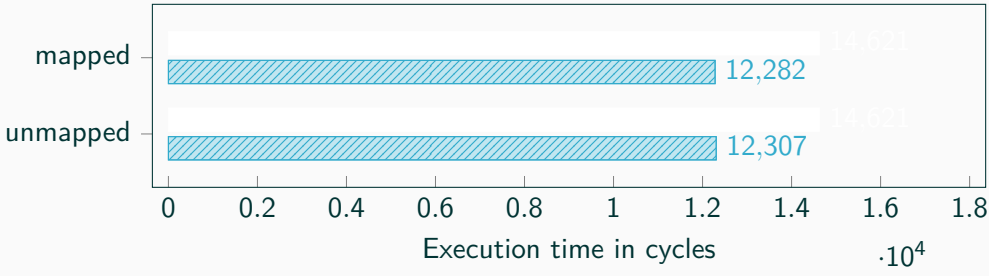
- Minimize the number of implicit TLB flushes
- x86 performs implicit TLB flush on every CR3 update
- PTE global bit
  - Preserve mappings that exist in every process
  - Excluded from implicit TLB flushes
- Disable the global bits
  - Negligible overhead

Solving Challenge 3

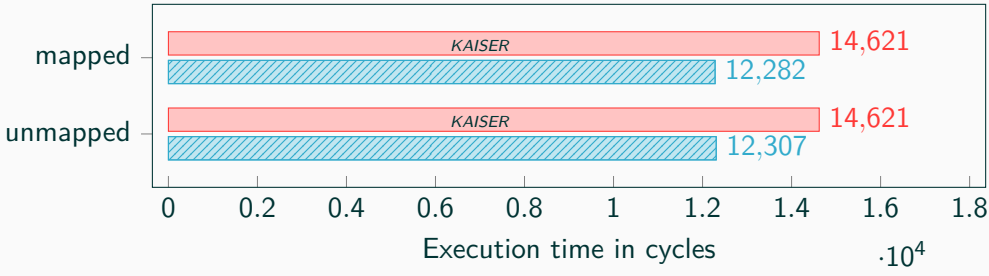
- Minimize the number of implicit TLB flushes
- x86 performs implicit TLB flush on every CR3 update
- PTE global bit
  - Preserve mappings that exist in every process
  - Excluded from implicit TLB flushes
- Disable the global bits
  - Negligible overhead
- Tag TLB with CR3 [Ven+12]

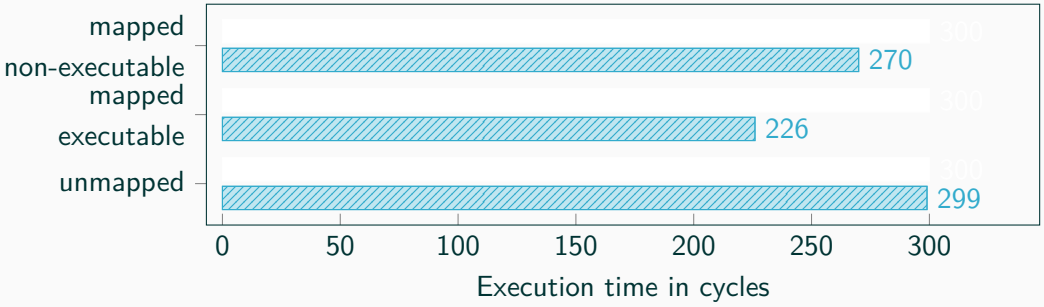
Evaluation

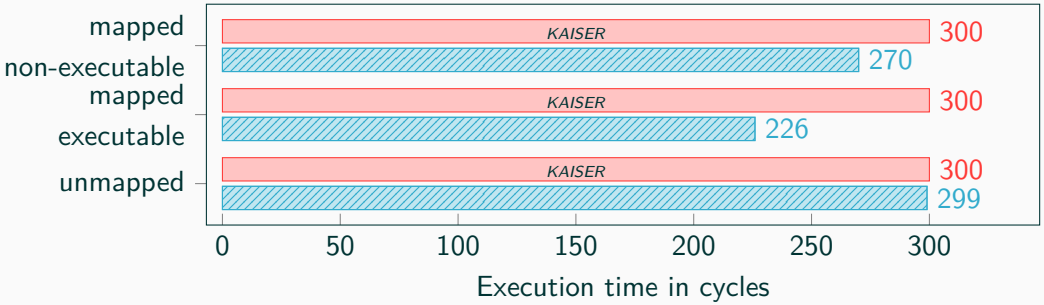
Double Page Fault Attack

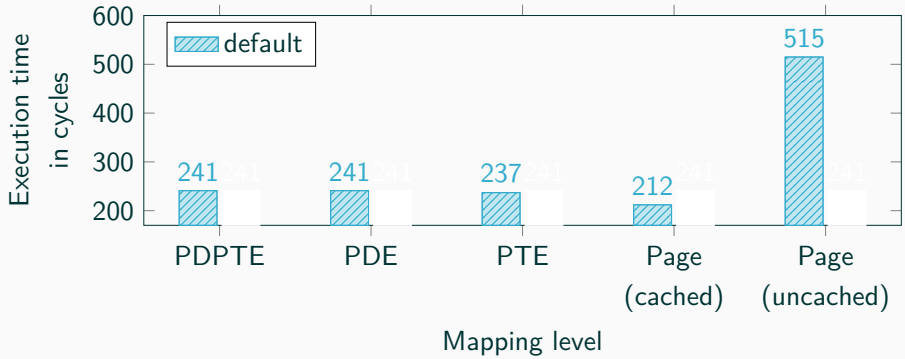


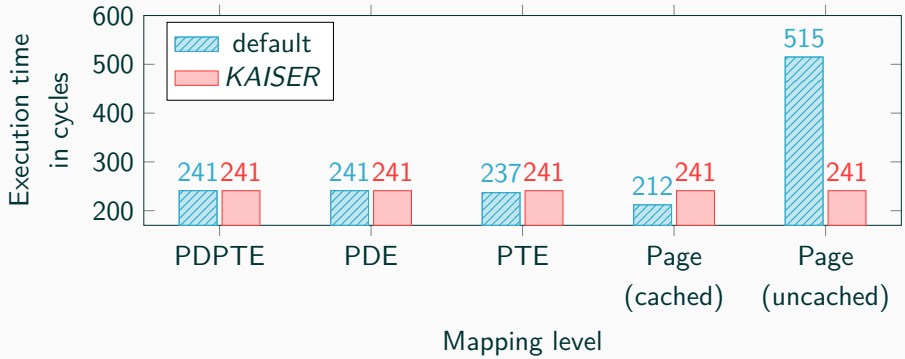
Double Page Fault Attack

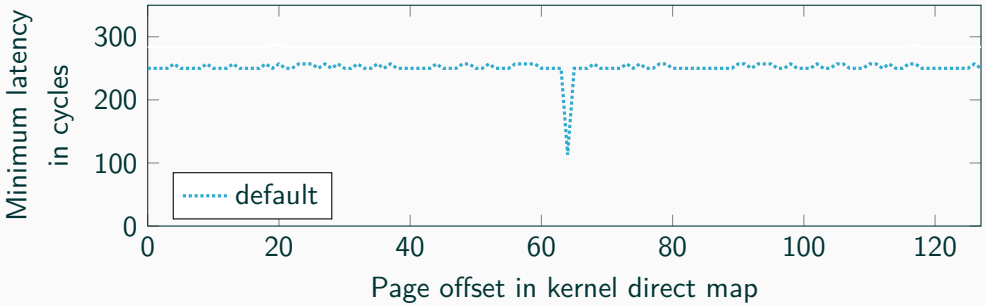


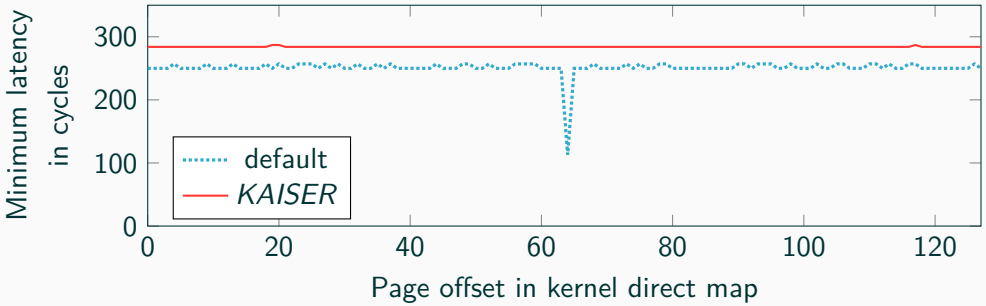

mapped
non-executable
mapped
executable
unmapped
300
270
300
226
300
299
0
50
100
150
200
250
300
Execution time in cycles


mapped
non-executable
mapped
executable
unmapped
KAISER
300
270
KAISER
300
226
KAISER
300
299
0
50
100
150
200
250
300
Execution time in cycles


Execution time in cycles
600
500
400
300
200
default
241
241
241
241
237
241
212
241
515
241
PDPTE
PDE
PTE
Page (cached)
Page (uncached)
Mapping level


Execution time in cycles
600
500
400
300
200
default
KAISER
241
241
241
241
237
241
212
241
515
241
PDPTE
PDE
PTE
Page (cached)
Page (uncached)
Mapping level


Minimum latency
in cycles
300
200
100
0
default
0
20
40
60
80
100
120
Page offset in kernel direct map


Minimum latency
in cycles
300
200
100
0
default
KAISER
0
20
40
60
80
100
120
Page offset in kernel direct map

Performance

Benchmark	Kernel	Runtime				Average Overhead
		1 core	*2 cores*	*4 cores*	*8 cores*	
PARSEC 3.0	default	27:56,0 s	14:56,3 s	8:35,6 s	7:05,1 s	0.37 %
	KAISER	28:00,2 s	14:58,9 s	8:36,9 s	7:08,0 s	
pgbench	default	3:22,3 s	3:21,9 s	3:21,7 s	3:53,5 s	0.39 %
	KAISER	3:23,4 s	3:22,5 s	3:22,3 s	3:54,7 s	
SPLASH-2X	default	17:38,4 s	10:47,7 s	7:10,4 s	6:05,3 s	0.09 %
	KAISER	17:42,6 s	10:48,5 s	7:10,8 s	6:05,7 s	

Github

Source available on Github:

https://github.com/iaik/kaiser

Conclusion

Conclusion

- KASLR is enabled by default in Linux

Conclusion

- KASLR is enabled by default in Linux
- KAISER prevents existing side-channel attacks
  - Double Page Fault Attacks
  - TSX-based side-channel attacks
  - Prefetch side-channel attacks

Conclusion

- KASLR is enabled by default in Linux
- KAISER prevents existing side-channel attacks
  - Double Page Fault Attacks
  - TSX-based side-channel attacks
  - Prefetch side-channel attacks
- Minor performance overhead on modern commodity hardware

KASLR is Dead: Long Live KASLR

D. Gruss, M. Lipp, M. Schwarz, R. Fellner, C. Maurice, S. Mangard

July 5, 2017—ESSoS'17

Graz University of Technology

References

D. Gruss, C. Maurice, A. Fogh, M. Lipp, and S. Mangard. “Prefetch Side-Channel Attacks: Bypassing SMAP and Kernel ASLR”. In: *CCS’16*. 2016.

R. Hund, C. Willems, and T. Holz. “Practical Timing Side Channel Attacks against Kernel Space ASLR”. In: *S&P’13*. 2013.

Y. Jang, S. Lee, and T. Kim. “Breaking Kernel Address Space Layout Randomization with Intel TSX”. In: *CCS’16*. 2016.

G. Venkatasubramanian, R. J. Figueiredo, R. Illikkal, and D. Newell. “TMT: A TLB Tag Management Framework for Virtualized Platforms”. In: *International Journal of Parallel Programming* 40.3 (2012).