Prefetch Side-Channel Attacks: Bypassing SMAP and Kernel ASLR

D. Gruss, C. Maurice, A. Fogh†, M. Lipp, S. Mangard
Graz University of Technology, † G DATA Advanced Analytics

October 25, 2016 — CCS’16
Overview

- prefetch instructions don’t check privileges
- prefetch instructions leak timing information
Overview

- prefetch instructions don’t check privileges
- prefetch instructions leak timing information

exploit this to:

- locate a driver in kernel = defeat KASLR
- translate virtual to physical addresses
Intel being overspecific

NOTE

Using the PREFETCH instruction is recommended only if data does not fit in cache. Use of software prefetch should be limited to memory addresses that are managed or owned within the application context. Prefetching to addresses that are not mapped to physical pages can experience non-deterministic performance penalty. For example, specifying a NULL pointer (0L) as address for a prefetch can cause long delays.
Intel being overspecific

NOTE

Using the PREFETCH instruction is recommended only if data does not fit in cache.
Intel being overspecific

NOTE

Using the PREFETCH instruction is recommended only if data does not fit in cache. Use of software prefetch should be limited to memory addresses that are managed or owned within the application context.
Intel being overspecific

NOTE

Using the PREFETCH instruction is recommended only if data does not fit in cache. Use of software prefetch should be limited to memory addresses that are managed or owned within the application context. Prefetching to addresses that are not mapped to physical pages can experience non-deterministic performance penalty.
Intel being overspecific

NOTE

Using the PREFETCH instruction is recommended only if data does not fit in cache. Use of software prefetch should be limited to memory addresses that are managed or owned within the application context. Prefetching to addresses that are not mapped to physical pages can experience non-deterministic performance penalty. For example specifying a NULL pointer (0L) as address for a prefetch can cause long delays.
CPU Caches

Memory (DRAM) is slow compared to the CPU

- buffer frequently used memory
- every memory reference goes through the cache
- based on physical addresses
Memory Access Latency

- Cache hits
- Cache misses

Access time in cycles

Number of accesses
Unprivileged cache maintenance

Optimize cache usage:

- **prefetch**: suggest CPU to load data into cache
- **clflush**: throw out data from all caches

... based on *virtual* addresses
Software prefetching

`prefetch` instructions are somewhat unusual

- hints – can be ignored by the CPU
- do not check privileges or cause exceptions

but they do need to translate virtual to physical
Kernel must be mapped in every address space

Today’s operating systems:

Shared address space

User memory

Kernel memory

context switch
Address translation on x86-64

48-bit virtual address

PML4I (9 b) | PDPTI (9 b) | PDI (9 b) | PTI (9 b) | Offset (12 b)

4 KiB Page

 Byte 0
 Byte 1
 Offset
 Byte 4095
Address Translation Caches

Core 0
- ITLB
- DTLB
- PDE cache
- PDPTTE cache
- PML4E cache

Core 1
- ITLB
- DTLB
- PDE cache
- PDPTTE cache
- PML4E cache

Page table structures in system memory (DRAM)

Lookup direction
Address Space Layout Randomization (ASLR)

Process A
0

Process B
0

Process C
0
Address Space Layout Randomization (ASLR)

Same library – different offset!
Kernel Address Space Layout Randomization (KASLR)

Same driver – different offset!

Daniel Gruss, Graz University of Technology
October 25, 2016 — CCS’16
Kernel Address Space Layout Randomization (KASLR)

Same driver – different offset!
Kernel direct-physical map

Virtual address space

User

Kernel

Physical memory

max. phys.

0

0

$2^{47}$

$-2^{47}$

$-1$
Kernel direct-physical map

OS X, Linux, BSD, Xen PVM (Amazon EC2)
Translation-Level Oracle

Execution time in cycles

<table>
<thead>
<tr>
<th>Mapping level</th>
<th>PDPT</th>
<th>PD</th>
<th>PT</th>
<th>cached P.</th>
<th>uncached P.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>230</td>
<td>246</td>
<td>222</td>
<td>181</td>
<td>383</td>
</tr>
</tbody>
</table>
Address-Translation Oracle

User space

Kernel space

Cache
Address-Translation Oracle
Address-Translation Oracle

- User space
- Cache
- Kernel space

clflush → flush
Address-Translation Oracle

User space

Cache

Kernel space

prefetch

load
Address-Translation Oracle

User space

Kernel space

Cache

load

reload (cache hit)
Timing the prefetch instruction

The CPU may reorder prefetch instruction – a look at $\texttt{rdtscp}$
Timing the prefetch instruction

The CPU may reorder prefetch instruction – a look at rdtscp
Timing the prefetch instruction

The CPU may reorder prefetch instruction – a look at `rdtscp`
Timing the prefetch instruction

The CPU may reorder instructions

- instruction 1
- cpuid
- instruction 2
- cpuid
- instruction 3

but not over cpuid!
Windows 10 Memory layout

- HAL, kernel, kernel drivers located bewetween
  - start: 0xffff 8000 0000 0000
  - end : 0xffff 9fff ffff ffff
Breaking Windows KASLR

for all mapped pages (found via translation-level oracle):
Breaking Windows KASLR

for all mapped pages (found via translation-level oracle):
1. evict translation caches: `Sleep()` / access large memory buffer
Breaking Windows KASLR

for all mapped pages (found via translation-level oracle):

1. **evict** translation caches: `Sleep()` / access large memory buffer
2. perform **syscall** to driver
Breaking Windows KASLR

for all mapped pages (found via translation-level oracle):
1. evict translation caches: `Sleep()` / access large memory buffer
2. perform `syscall` to driver
3. time `prefetch(page address)`
Breaking Windows KASLR

for all mapped pages (found via translation-level oracle):

1. evict translation caches: `Sleep()` / access large memory buffer
2. perform syscall to driver
3. time `prefetch(page address)`
   → Fastest average access time is a driver page.
Breaking Windows KASLR

for all mapped pages (found via translation-level oracle):

1. evict translation caches: `Sleep()` / access large memory buffer
2. perform syscall to driver
3. time `prefetch(page address)`
   → Fastest average access time is a driver page.

Full attack on Windows 10 in < 12 seconds
Locate Kernel Driver (defeat KASLR)

Avg. exec. time [cycles]

Page offset in kernel driver region
Kernel exploits (10 years ago)

- overwrite return address
  → jump to userspace code
- overwrite stack pointer
  → switch to userspace stack
Mitigating kernel exploits

- Jump to userspace code? Nope! Hardware prevents that.
  - supervisor-mode execution prevention (SMEP)
- Switch to userspace stack? Nope! Hardware prevents that.
  - supervisor-mode access prevention (SMAP)
Kernel direct-physical map
Evading the mitigation

- get direct-physical-map address of userspace address
  → jump/switch there

known as “ret2dir” attacks (Kemerlis et al. 2014)
Mitigating the evasion

- getting rid of direct-physical map?
Mitigating the evasion

- getting rid of direct-physical map? Apparently not.
  → do not leak physical addresses to user
Prefetching via direct-physical map

```
Page offset in direct-physical map

Min. acc. latency [cycles]

Daniel Gruss, Graz University of Technology
October 25, 2016 — CCS'16
```
Prefetching via direct-physical map

- immediately leaks a direct-physical map address
  → no physical address necessary (compared to ret2dir)
- if direct-physical map offset is known
  → leaks physical address
Countermeasure

Today’s operating systems:

Shared address space

User memory \[\rightarrow\] Kernel memory

0 \[\rightarrow\] −1

context switch

Stronger kernel isolation:

User address space

User memory \[\rightarrow\] Not mapped

0 \[\rightarrow\] −1

context switch

Kernel address space

Not mapped \[\rightarrow\] Kernel memory

0 \[\rightarrow\] −1

Interrupt dispatcher
Conclusion

- prefetch leaks significant information
- we can locate a driver in the kernel and thus break KASLR
- break SMAP/SMEP and get physical addresses
- countermeasure could be implemented in software
Prefetch Side-Channel Attacks: Bypassing SMAP and Kernel ASLR

D. Gruss, C. Maurice, A. Fogh†, M. Lipp, S. Mangard
Graz University of Technology, † G DATA Advanced Analytics

October 25, 2016 — CCS’16