Using Undocumented CPU Behaviour to See into Kernel Mode and Break KASLR in the Process

Anders Fogh and Daniel Gruss

August 4, 2016
About this presentation

This talk is about a class of microarchitectorial attacks

► Not about software bugs
► It is about CPU design as an attack vector
► But not about Instruction Set Architecture
► Focus on Intel x86-64 - applies to other architectures too
Take aways

- CPU design is security relevant
- Prefetch instructions leak information
Take aways

- CPU design is security relevant
- Prefetch instructions leak information

Exploit this to:
- Locate a driver in kernel = defeat KASLR
- Translate Virtual to physical addresses for other attacks
Introduction

Memory subsystem

Kernel Address-space Layout Randomization (KASLR)

Prefetch Side Channel

Prefetching the Kernel

Case study: Defeating Windows 7 KASLR

Case study: Exploiting direct-physical maps

Bonus material
Introduction
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.
The chamber of secrets

NOTE

Using the PREFETCH instruction is recommended only if data does not fit in cache.
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.
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.
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.
The chamber of secrets (translation)

PLEASE
The chamber of secrets (translation)

PLEASE,

only use prefetch as Intel intends
The chamber of secrets (translation)

PLEASE,

only use prefetch as Intel intends,
or else Intel will be angry
The chamber of secrets (translation)

PLEASE,

only use prefetch as Intel intends,
or else Intel will be angry,
and there is no reason why anyone would measure the execution time.
SHOULDN'T HAVE SAID THAT

I SHOULD NOT HAVE SAID THAT
Whoami

- Anders Fogh
- Principal Security Researcher, GDATA Advanced Analytics
- Playing with malware since 1992
- Twitter: @anders_fogh
- Email: anders.fogh@gdata-adan.de
Whoami

- Daniel Gruss
- PhD Student, Graz University of Technology
- Currently intern at Microsoft Research Cambridge
- Twitter: @lavados
- Email: daniel.gruss@iaik.tugraz.at
And the team

The research team

- Clémentine Maurice
- Moritz Lipp
- Stefan Mangard

from Graz University of Technology
Memory subsystem
CPU Caches

Memory (DRAM) is slow compared to the CPU
- buffer frequently used memory for the CPU
- every memory reference goes through the cache
- transparent to OS and programs
Memory Access Latency

- **Access time in cycles**
  - 50
  - 100
  - 150
  - 200
  - 250
  - 300
  - 350
  - 400

- **Number of accesses**
  - Cache hits
  - Cache misses

**Graph Details**

- **Y-axis**: Number of accesses
- **X-axis**: Access time in cycles
- **Legend**:
  - cache hits
  - cache misses
Data Caches

- core 0
  - L1
  - L2
- core 1
  - L1
  - L2
- core 2
  - L1
  - L2
- core 3
  - L1
  - L2

- LLC slice 0
- LLC slice 1
- LLC slice 2
- LLC slice 3

Last-level cache:
- shared memory shared is in cache, across cores!
- physically indexed
- need physical address to manipulate
- only one cache entry per physical address
Unprivileged cache maintainance

User programs can optimize cache usage:
- **prefetch**: suggest CPU to load data into cache
- **clflush**: throw out data from all caches

... based on virtual addresses
Caches: Software control

There are 5 `prefetch` instructions:

- `prefetcht0`: suggest CPU to load data into L1
- `prefetcht1`: suggest CPU to load data into L2
- `prefetcht2`: suggest CPU to load data into L3
- `prefetchnta`: suggest CPU to load data for non-temporal access
- `prefetchw`: suggest CPU to load data with intention to write

Actual behaviour varies between CPU models
Caches: Software control

The `prefetch` instructions are somewhat unusual

- Hints – can be ignored by the CPU
- Do not check privileges or cause exceptions
Virtual and physical addressing

Why address translation: Run multiple processes securely on a single CPU
- Let applications run in their own virtual address space
- Create exchangeable map from “virtual memory” to “physical memory”
- Privileges are checked on memory accesses
- Managed by the operating system kernel
Address translation on x86-64

48-bit virtual address

- PML4I (9 b)
- PDPTI (9 b)
- PDI (9 b)
- PTI (9 b)
- Offset (12 b)
Problem: translation tables are stored in physical memory
Solution: Address Translation Caches

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

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

Page table structures in system memory (DRAM)
Kernel Address-space Layout Randomization (KASLR)
Kernel is mapped in every process

Today’s operating systems:

Shared address space

User memory

Kernel memory

context switch
Address-Space Layout Randomization (ASLR)

- Kernel and drivers at randomized offsets in virtual memory
- Mitigates code reuse attacks e.g. return-oriented-programming
- Attacks based on read primitives or write primitives
Address-Space Layout Randomization (ASLR)

- Kernel and drivers at randomized offsets in virtual memory
- Mitigates code reuse attacks e.g. return-oriented-programming
- Attacks based on read primitives or write primitives
- But: leaking kernel/driver addresses defeats ASLR
Kernel direct-physical map

Available on many operating systems / hypervisors
- OS X
- Linux
- BSD
- Xen PVM (Amazon EC2)

But not on Windows!
Kernel direct-physical map

Available on many operating systems / hypervisors

- OS X
- Linux
- BSD
- Xen PVM (Amazon EC2)

But not on Windows!
Prefetch Side Channel
Summary

1. The kernel is mapped in every process

The prefetch instruction takes a virtual address as input. To manipulate L3 a physical address is needed. The prefetch instruction must translate.

The prefetch instruction does not check privileges. Any address can be prefetched.

Translation is cached. Lookup searches caches in a fixed order. Can we measure a time difference?
Summary

1. The kernel is mapped in every process

2. The `prefetch` instruction takes a virtual address as input. To manipulate L3 a physical address is needed. The prefetch instruction must translate.
Summary

1. The kernel is mapped in every process

2. The `prefetch` instruction takes a virtual address as input
   - To manipulate L3 a physical address is needed
   - The `prefetch` instruction must translate

3. The `prefetch` instruction does not check privileges
   - Any address can be prefetched
Summary

1. The kernel is mapped in every process

2. The `prefetch` instruction takes a virtual address as input
   - To manipulate L3 a physical address is needed
   = The `prefetch` instruction must translate

3. The `prefetch` instruction does not check privileges
   = Any address can be prefetched

4. Translation is cached
   - Lookup searches caches in a fixed order
Summary

1. ▶ The kernel is mapped in every process

2. ▶ The `prefetch` instruction takes a virtual address as input
   ▶ To manipulate L3 a physical address is needed
   = The prefetch instruction must translate

3. ▶ The `prefetch` instruction does not check privileges
   = Any address can be prefetched

4. ▶ Translation is cached
   ▶ Lookup searches caches in a fixed order

Can we measure a time difference?
There is a timing difference!

Idea: Would this also work on inaccessible kernel memory?
Prefetching the Kernel
Definition of our translation oracle

- Timing the `prefetch` instruction on an arbitrary address will recover the translation level
Recovering a map of the kernel

Recovering /proc/pid/pagemap in 4 steps:

1. Recover mappings in PML4 by using the translation oracle on kernel addresses
Recovering a map of the kernel

Recovering `/proc/pid/pagemap` in 4 steps:

1. Recover mappings in PML4 by using the translation oracle on kernel addresses
2. Recover mappings in PDPT by using the translation oracle on kernel addresses
Recovering a map of the kernel

Recovering `/proc/pid/pagemap` in 4 steps:

1. Recover mappings in PML4 by using the translation oracle on kernel addresses
2. Recover mappings in PDPT by using the translation oracle on kernel addresses
3. Recover mappings in PD by using the translation oracle on kernel addresses

Complete process takes from seconds to hours depending on how pages are actually mapped by the operating system.
Recovering a map of the kernel

Recovering `/proc/pid/pagemap` in 4 steps:

1. Recover mappings in PML4 by using the translation oracle on kernel addresses
2. Recover mappings in PDPT by using the translation oracle on kernel addresses
3. Recover mappings in PD by using the translation oracle on kernel addresses
4. Recover mappings in PT by using the translation oracle on kernel addresses

Complete process takes from seconds to hours depending on how pages are actually mapped by the operating system.
Recovering a map of the kernel

Recovering `/proc/pid/pagemap` in 4 steps:

1. Recover mappings in PML4 by using the translation oracle on kernel addresses
2. Recover mappings in PDPT by using the translation oracle on kernel addresses
3. Recover mappings in PD by using the translation oracle on kernel addresses
4. Recover mappings in PT by using the translation oracle on kernel addresses

Complete process takes from seconds to hours depending on how pages are actually mapped by the operating system.
The address-translation oracle

The address-translation oracle tell us whether virtual address $p$ and $\bar{p}$ map to the same physical address

1. Use `clflush` to remove $p$ from cache
The address-translation oracle

The address-translation oracle tell us whether virtual address $p$ and $\bar{p}$ map to the same physical address

1. Use `clflush` to remove $p$ from cache
2. Use `prefetch` to load $\bar{p}$ into cache

Beware! `prefetch` is a hint!
The address-translation oracle

The address-translation oracle tell us whether virtual address $p$ and $\bar{p}$ map to the same physical address

1. Use `clflush` to remove $p$ from cache
2. Use `prefetch` to load $\bar{p}$ into cache
3. time access of $p$. If fast it was cached: $p$: maps to the same physical memory as $\bar{p}$

Beware! `prefetch` is a hint!
Timing instructions

The CPU may reorder instructions – a look at rdtscp

- instruction 1
- rdtscp
- instruction 2
- rdtscp
- instruction 3
Timing instructions

The CPU may reorder instructions – a look at rdtscp
Timing instructions

The CPU may reorder instructions – a look at `rdtscp`

- Instruction 1
- `rdtscp`
- Instruction 2
- `rdtscp`
- Instruction 3
Timing instructions

The CPU may reorder instructions – a look at `mfence`

- instruction 1
- `mfence`
- instruction 2
- `mfence`
- instruction 3
Timing instructions

The CPU may reorder instructions – a look at `mfence`

```
instruction 1
```
```
mfence
```
```
instruction 2
```
```
mfence
```
```
instruction 3
```
Timing instructions

The CPU may reorder instructions – a look at mfence

```
instruction 1

mfence

instruction 2

mfence

instruction 3
```
Timing instructions

The CPU may reorder instructions

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

but not over cpuid!
Timing the prefetch instruction

The CPU may reorder prefetch instruction – a look at rdtscp

```
prefetch

rdtscp

prefetch

rdtscp

prefetch
```
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

```
prefetch
rdtscp
prefetch
rdtscp
prefetch
```
Timing the prefetch instruction

The CPU may reorder instructions – a look at mfence

- prefetch
- mfence
- prefetch
- mfence
- prefetch
Timing the prefetch instruction

The CPU may reorder instructions – a look at mfence

```
prefetch

mfence

prefetch

mfence

prefetch
```
Timing the prefetch instruction

The CPU may reorder instructions – a look at mfence
Timing instructions

Combine instructions in clever procedures to
▶ prevent reordering of prefetch if necessary
▶ avoid noise from `cpuid` if possible
Combine instructions in clever procedures to

- prevent reordering of prefetch if necessary
  
  \begin{verbatim}
  = mfence rdtscp cpuid target instruction cpuid rdtscp mfence
  \end{verbatim}

- avoid noise from `cpuid` if possible
Timing instructions

Combine instructions in clever procedures to

- prevent reordering of prefetch if necessary
  
  = mfence rdtscp cpuid target instruction cpuid rdtscp mfence

- avoid noise from cpuid if possible
  
  = mfence cpuid rdtscp target instruction rdtscp cpuid mfence
Timing instructions

Combine instructions in clever procedures to

- prevent reordering of prefetch if necessary
  = mfence rdtscp cpuid \textcolor{red}{\textbf{target instruction}} cpuid rdtscp mfence

- avoid noise from \texttt{cpuid} if possible
  = mfence cpuid rdtscp \textcolor{red}{\textbf{target instruction}} rdtscp cpuid mfence

We use either the \texttt{prefetchnta} or \texttt{prefetcht2} instructions and a \texttt{mov} instruction for memory access
Case study: Defeating Windows 7 KASLR
Windows 7 Memory layout

- HAL and kernel located between:
  - start: 0xffff f800 0000 0000
  - end: 0xffff f87f ffff ffff

- Kernel drivers:
  - start: 0xffff f880 0000 0000
  - end: 0xffff f8ff ffff ffff
1. Map the drivers address space using translation recovery attack
Windows 7 Breaking KASLR

1. Map the drivers address space using translation recovery attack
2. Evict the page translation caches: `Sleep()` and/or access large memory buffer
Windows 7 Breaking KASLR

1. Map the drivers address space using translation recovery attack
2. Evict the page translation caches: `Sleep()` and/or access large memory buffer
3. Perform a syscall to targeted driver
Windows 7 Breaking KASLR

1. Map the drivers address space using translation recovery attack
2. Evict the page translation caches: `Sleep()` and/or access large memory buffer
3. Perform a syscall to targeted driver
4. Time `prefetch(PageAddress)`
Windows 7 Breaking KASLR

1. Map the drivers address space using translation recovery attack
2. Evict the page translation caches: `Sleep()` and/or access large memory buffer
3. Perform a syscall to targeted driver
4. Time `prefetch(PageAddress)`
5. Repeat 2,3,4 for all pages found in 1
   Fastest average access time is right address.
Locate Kernel Driver (defeat KASLR)
Case study: Exploiting direct-physical maps
Kernel exploits

- 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
Circumventing the mitigation

Prefetching via direct-physical map

▶ use known address or translation recovery attack to find the direct-physical-map
▶ find user mode address in with direct-physical map using address-translation oracle
Circumventing the mitigation

Prefetching via direct-physical map

![Graph showing min. access latency in cycles vs. page offset in direct-physical map. The graph indicates a drop in latency at a specific page offset.]
Prefetching via direct-physical map

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

- works on Linux
- works on OSX
- works on Xen PVM (on Amazon EC2)
- does not work on Windows
  - no direct-physical map ;)


Cache side-channel attacks

Powerful side channel in the cloud
- infer user input
- crypto key recovery
- cross-VM, cross-core, even cross-CPU
- any architecture
Cache mapping

- slice function $H$ unknown
- reverse-engineered by Hund et al. 2013; Maurice et al. 2015; Inci et al. 2015; Yarom et al. 2015

$\rightarrow$ we need the physical address
Rowhammer

Rowhammer: yet another attack requiring physical address information
- Rowhammer: bit flip at a random location in DRAM
- exploitable → gain root privileges Seaborn and Dullien 2015
Rowhammer

“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice

![DRAM bank diagram](image)

row buffer
Rowhammer

“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice
Rowhammer

“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice
Rowhammer

“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice

![Diagram of Rowhammer](image-url)
Rowhammer

“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice
Rowhammer

“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
...```

```
row buffer
```

```
bit flips in row 2!
```
How is DRAM organized?
How is DRAM organized?

channel 0

channel 1
How is DRAM organized?

channel 0

channel 1

back of DIMM: rank 1

front of DIMM: rank 0
How is DRAM organized?

- Channel 0:
  - Front of DIMM: rank 0
  - Back of DIMM: rank 1

- Channel 1:
  - Front of DIMM: rank 0
  - Chip
DRAM organization example

- bits in cells in rows
- access: activate row, copy to row buffer
- cells leak $\rightarrow$ refresh necessary
- cells leak faster upon proximate accesses
DRAM mapping functions on a DDR4 system

Again: based on physical addresses
Summary

- Defeat SMAP/SMEP through direct-physical map
- Leak physical addresses
  - Perform cache attacks
  - Perform Rowhammer attacks
CPU Design is security relevant (prefetch leaks significant information)
We can locate a driver in the kernel and thus break KASLR
We can break SMAP/SMEP and get physical addresses to assist other attacks
Using Undocumented CPU Behaviour to See into Kernel Mode and Break KASLR in the Process

Anders Fogh and Daniel Gruss

August 4, 2016
Bonus material
Stronger Kernel Isolation

Today’s operating systems:

Shared address space

User memory | Kernel memory
0 | -1

context switch

Stronger kernel isolation:

User address space

User memory | Not mapped
0 | -1

context switch

Kernel memory

Interrupt dispatcher

addr. space switch

Not mapped

0 | -1

Kernel address space
Bibliography I


Maurice, Clémentine et al. (2015). “Reverse Engineering Intel Complex Addressing Using Performance Counters”. In: RAID.
