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

Anders Fogh and Daniel Gruss

August 5, 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

Take aways

- CPU design is security relevant
- Prefetch instructions leak information

Take aways

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

The chamber of secrets

NOTE

The chamber of secrets

NOTE

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

The chamber of secrets

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.

The chamber of secrets

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.

The chamber of secrets

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,

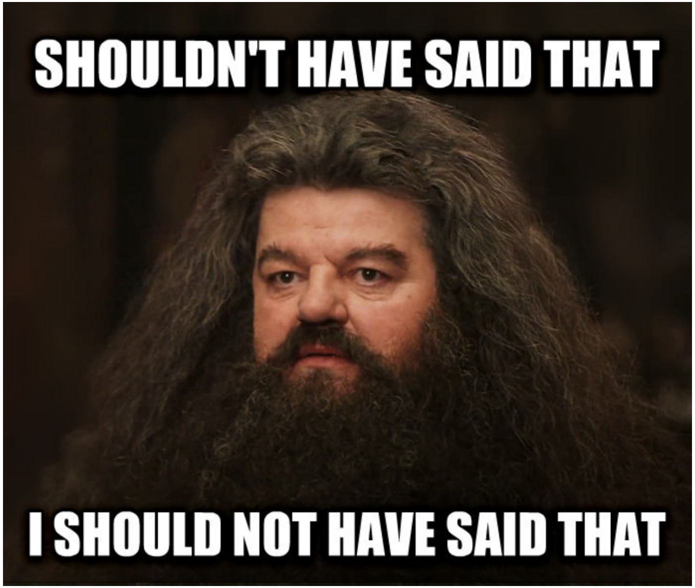
only use prefetch as Intel intends,
or else Intel will be angry

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.

The chamber of secrets (translation)



Whoami

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

GDATA | ADVANCED ANALYTICS

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

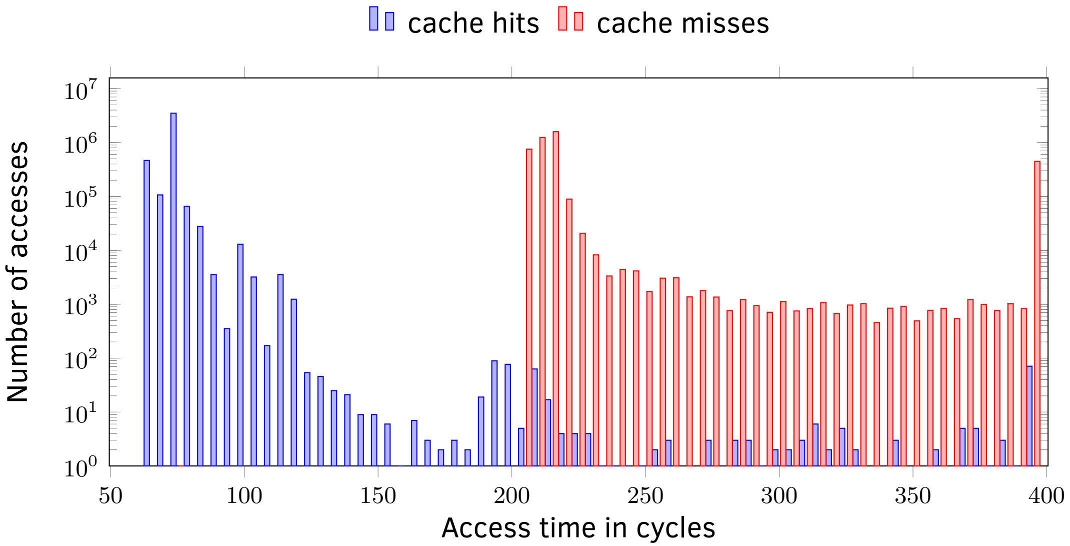
Memory subsystem

CPU Caches

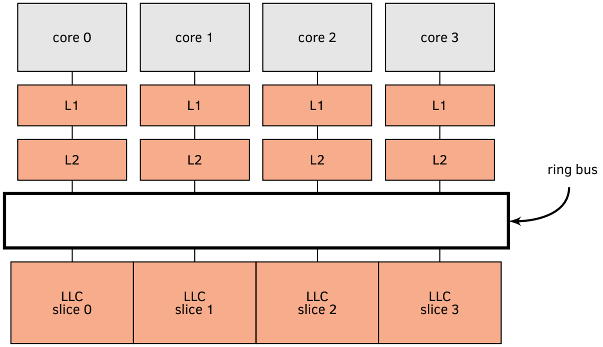
Memory (DRAM) is slow compared to the CPU

- buffer frequently used memory for the CPU
- every memory reference goes through the cache
- transparent to OS and programs

Memory Access Latency



Data Caches



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

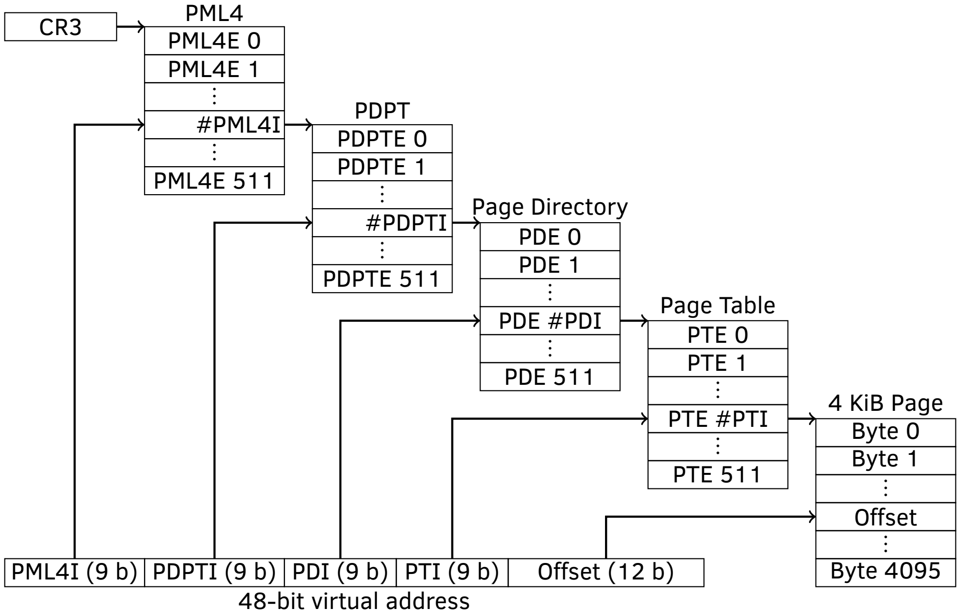
- Hints – can be ignored by the CPU
- Do not check privileges or cause exceptions

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

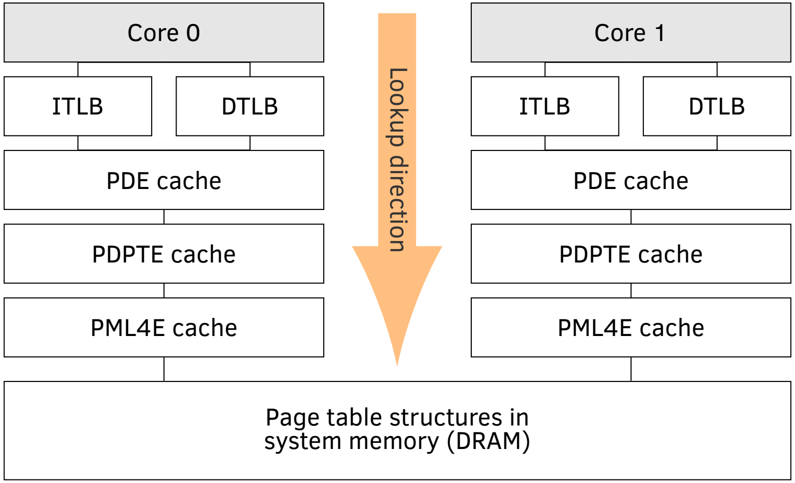
Address translation on x86-64



Address Translation Caches

Problem: translation tables are stored in physical memory

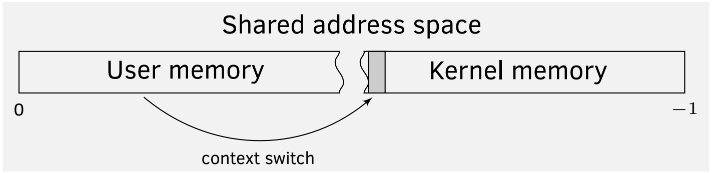
Solution: Address Translation Caches



Kernel Address-space Layout Randomization (KASLR)

Kernel is mapped in every process

Today's operating systems:



Address-Space Layout Randomization (ASLR)

- Code, Heap, Stack, ... are
- Mitigates code reuse attacks e.g. return-oriented-programming
- Attacks based on read primitives or write primitives

Address-Space Layout Randomization (ASLR)

- Code, Heap, Stack, ... are
- Mitigates code reuse attacks e.g. return-oriented-programming
- Attacks based on read primitives or write primitives
- But: leaking kernel/driver addresses defeats ASLR

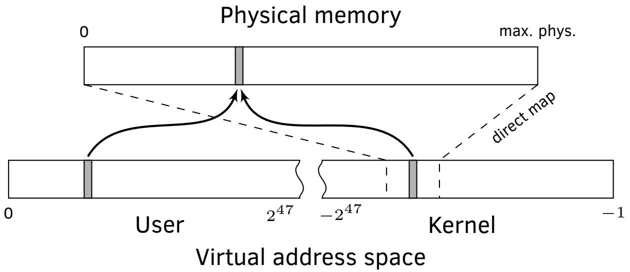
Address-Space Layout Randomization (ASLR)

- 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

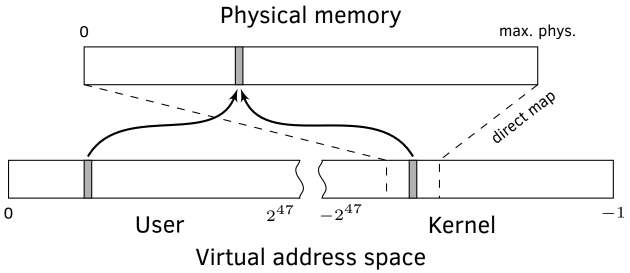
Address-Space Layout Randomization (ASLR)

- 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



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

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

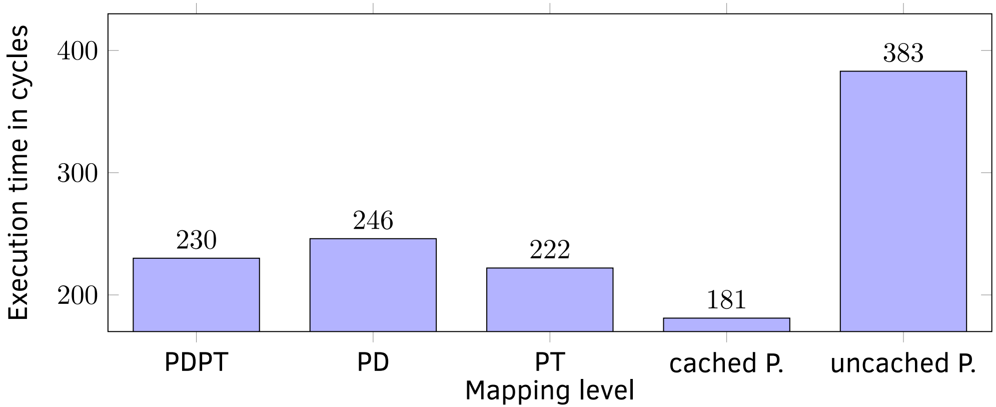
   = Any address can be prefetched
4.
   - Translation is cached
   - Lookup searches caches in a fixed order

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

A translation oracle

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

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

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

The address-translation oracle

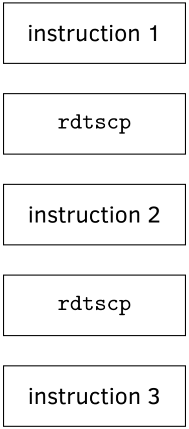
The address-translation oracle tell us whether virtual address p and $\bar{p}$ map to the same physical address

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!

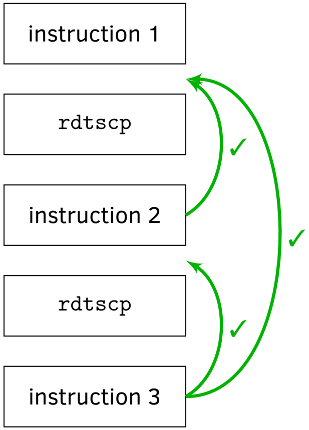
Timing instructions

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



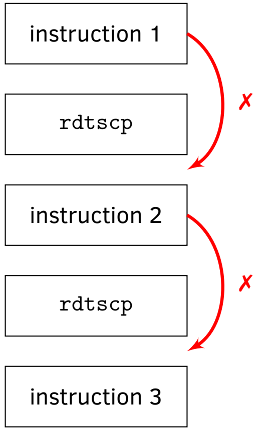
Timing instructions

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



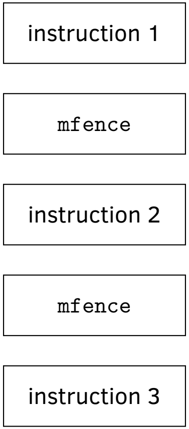
Timing instructions

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



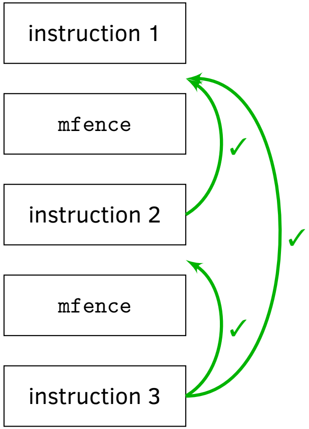
Timing instructions

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



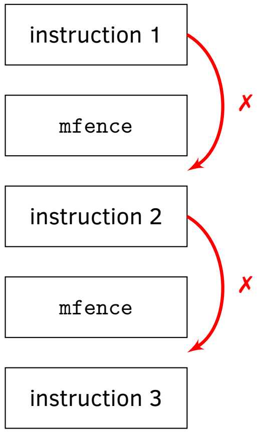
Timing instructions

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



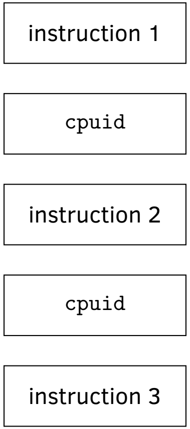
Timing instructions

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



Timing instructions

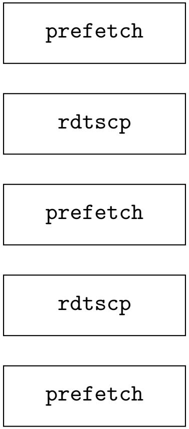
The CPU may reorder instructions



but not over `cpuid`!

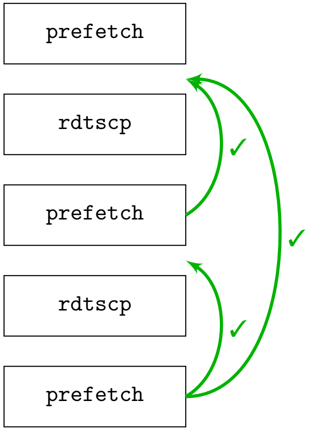
Timing the prefetch instruction

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



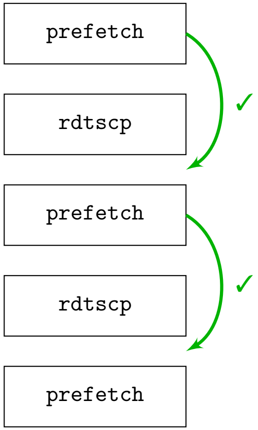
Timing the prefetch instruction

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



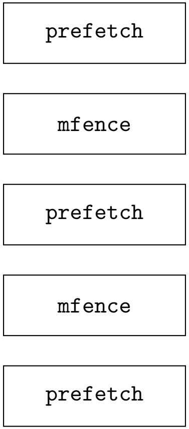
Timing the prefetch instruction

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



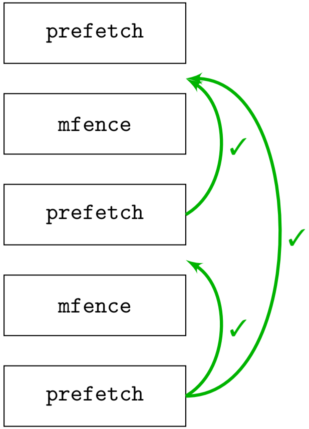
Timing the prefetch instruction

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



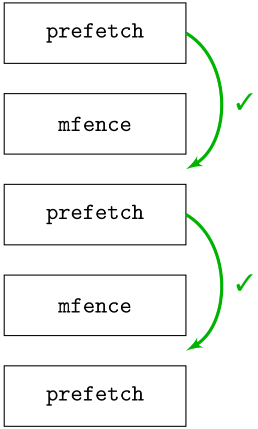
Timing the prefetch instruction

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



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

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

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

We use either the `prefetchnta` or `prefetcht2` instructions and a `mov` instruction for memory access

Case study: Defeating Windows 7 KASLR

Windows 7 Memory layout

- HAL and kernel located bewetween
  - start: `0xffff f800 0000 0000`
  - end : `0xffff f87f ffff ffff`
- Kernel drivers
  - start: `0xffff f880 0000 0000`
  - end : `0xffff f8ff ffff ffff`

Windows 7 Breaking KASLR

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

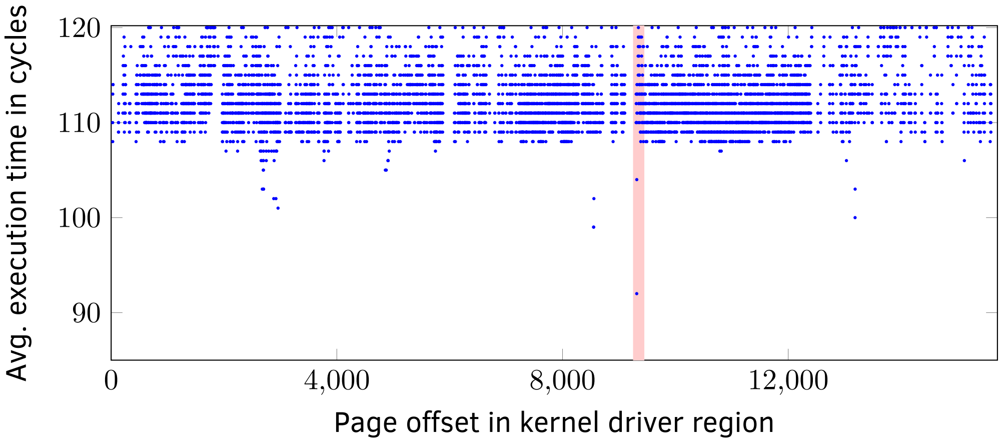
1. Map the drivers address space using translation recovery attack
2. Evict the page translation caches: `Sleep()` and/or access large memory buffer
3. Perform a syscall to targeted driver
4. Time `prefetch(PageAddress)`

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

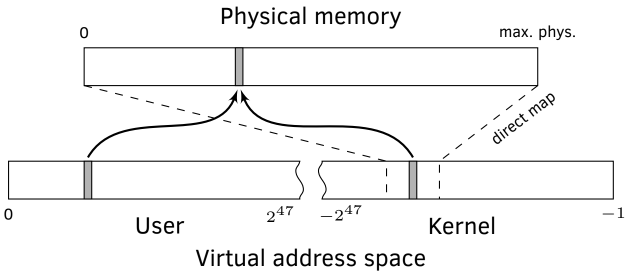
- Jump to userspace code? Nope! Hardware prevents that.

= 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

$\rightarrow$ jump/switch there

Known as “ret2dir” attacks **Kemerlis2014**

Mitigating the evasion

- Getting rid of direct-physical map?

Mitigating the evasion

- Getting rid of direct-physical map?Apparently not.

$\rightarrow$ Do not leak kernel addresses to user

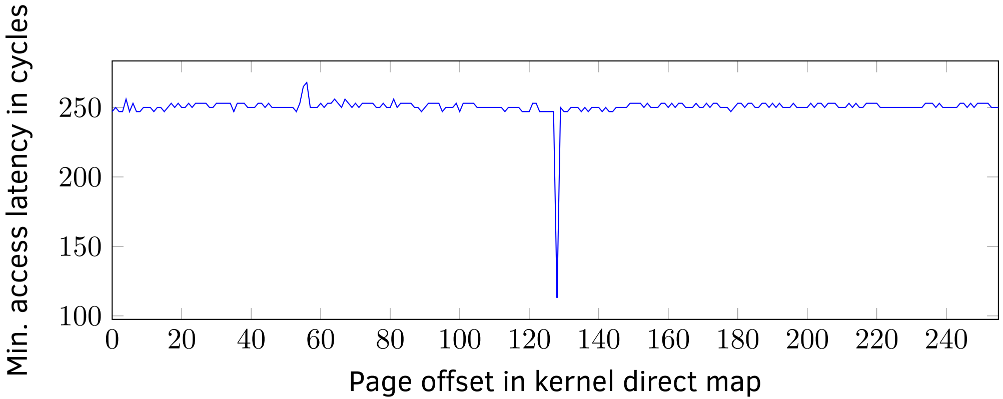
Circumventing the mitigation

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



Prefetching via direct-physical map

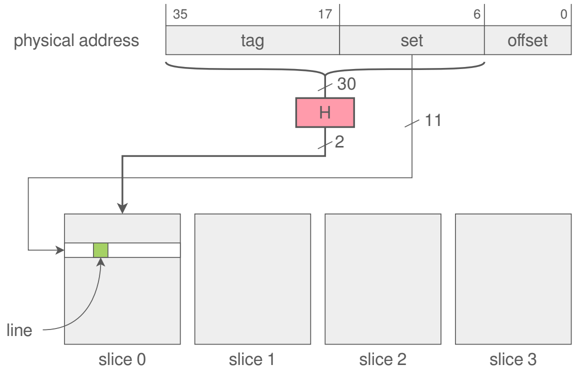
- ▸ immediately leaks a direct-physical map address
- → no information leak necessary (compared to ret2dir)
- ▸ if direct-physical map offset is known
- → leaks physical address

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 **Hund2013**; **Maurice2015RAID**; **Inci2015iacr**; **Yarom2015iacr**

$\rightarrow$ we need the physical address

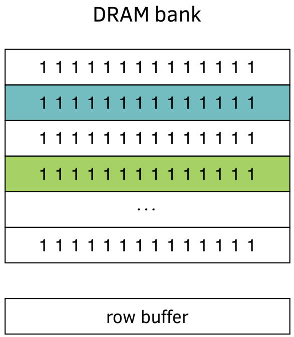
Rowhammer

Rowhammer: yet another attack requiring physical address information

- Rowhammer: bit flip at a random location in DRAM
- exploitable $\rightarrow$ gain root privileges **SeabornBlackhat2015**

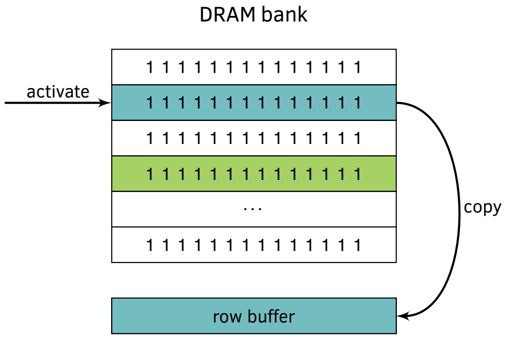
Rowhammer

"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



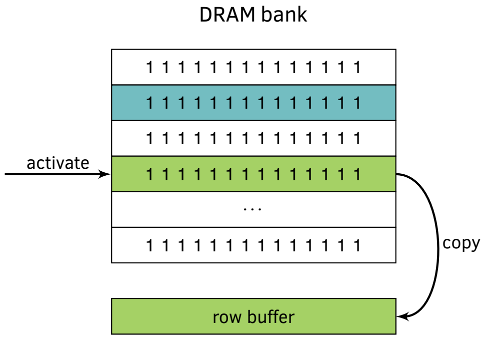
Rowhammer

"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



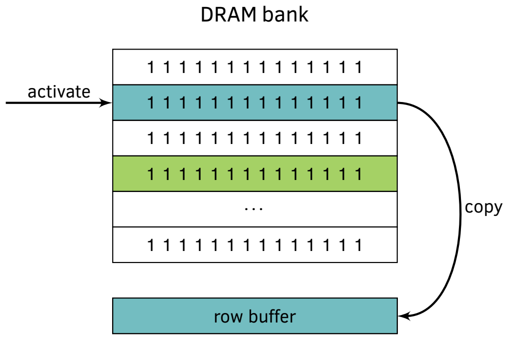
Rowhammer

"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



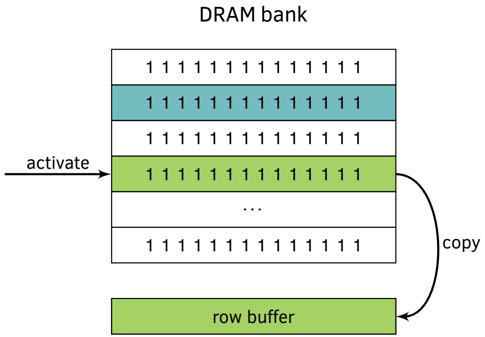
Rowhammer

“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



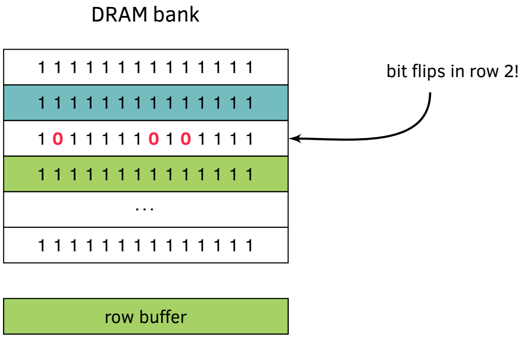
Rowhammer

“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



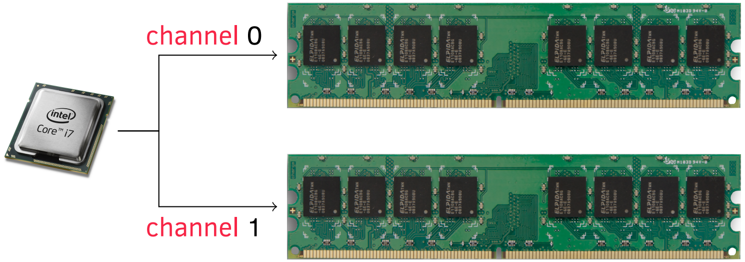
Rowhammer

"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

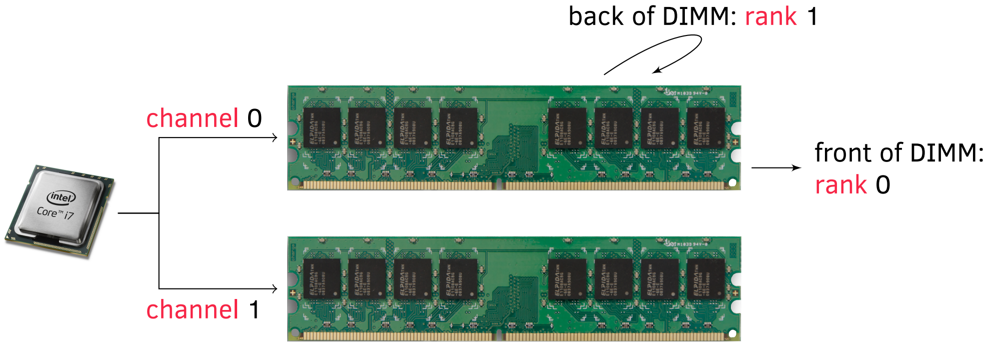


How is DRAM organized?

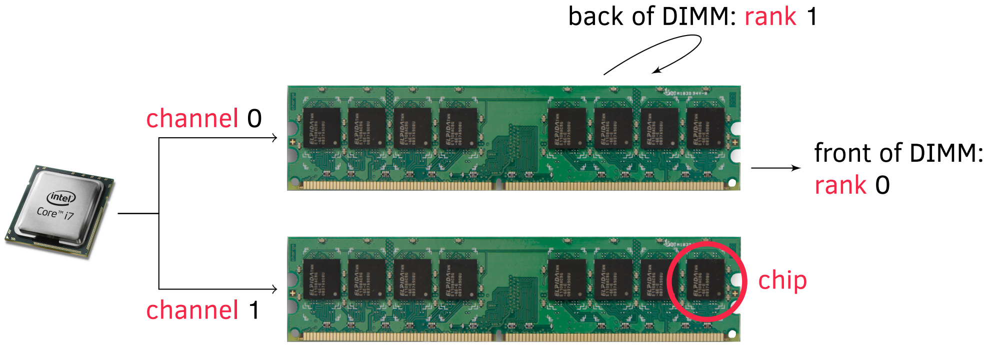
How is DRAM organized?



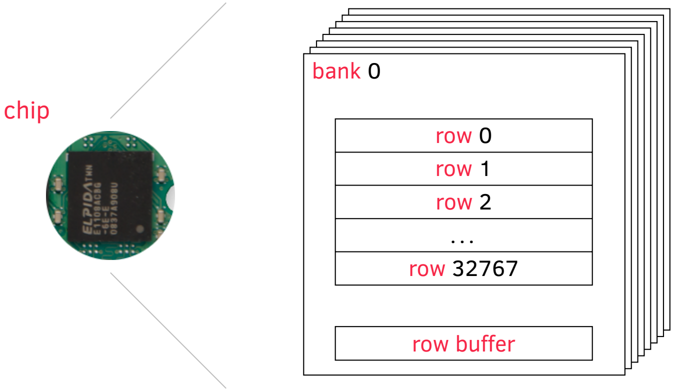
How is DRAM organized?



How is DRAM organized?

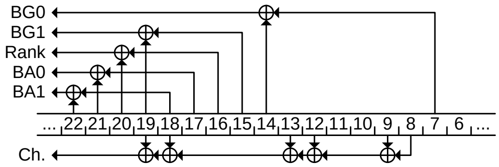


DRAM organization example



- 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

Black Hat Sound Bytes.

- 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

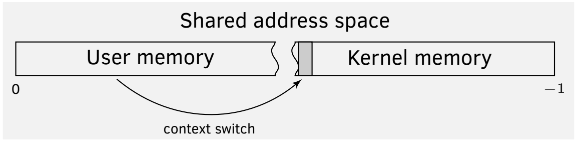
Anders Fogh and Daniel Gruss

August 5, 2016

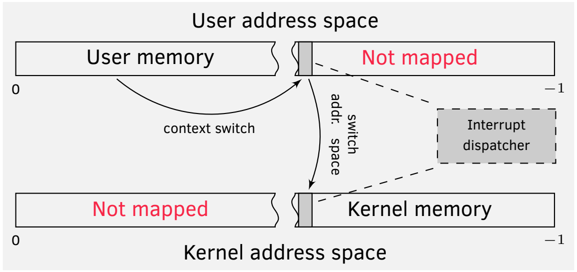
Bonus material

Stronger Kernel Isolation

Today's operating systems:



Stronger kernel isolation:



Bibliography I