

The Bad Neighbor

Out-of-Order Execution and Its
Applications

Sophia d'Antoine
November 7th, 2017

**TRAIL
OF BITS**

whoami

Masters in CS from RPI

- Exploiting Intel's CPU pipelines

Work at Trail of Bits

- Senior Security Researcher
- Program Analysis / Ethereum Smart Contracts

DEFCON (CTF), CSAW

Stats

- 12 Conferences Worldwide
- 3 Program Committees
- 2 Security Panels
- 1 Paper Published
- 1 Keynote



Side Channels

Hardware Side Channels in Virtualized
Environments

TRAIL
OF
BITS

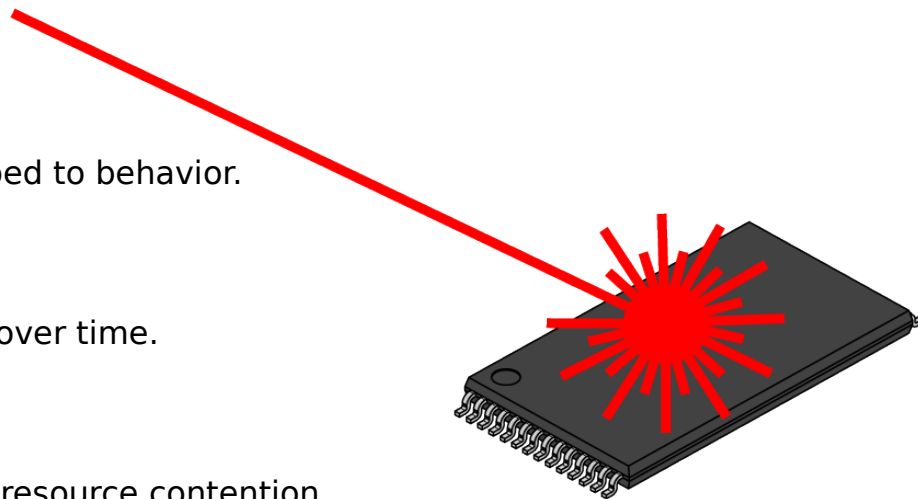
What are side channel attacks?

- Attacker can observe the target system. Must be 'neighboring' or co-located.
- Ability to repeatedly query the system for leaked artifacts.
- Artifacts: changes in how a process interacts with the computer

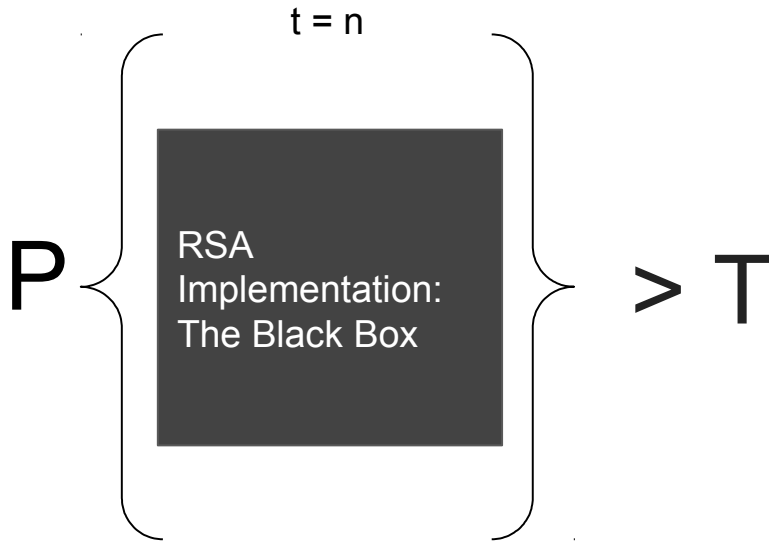
Variety of Side Channels

Different target systems implies different methods for observing.

- Fault attacks
 - Requires access to the hardware.
- Simple power analysis
 - Requires proximity to the system.
 - Power consumption measurement mapped to behavior.
- Different power analysis
 - Requires proximity to the system.
 - Statistics and error correction gathered over time.
- Timing attacks
 - Requires same process co-location.
 - Network packet delivery, cache misses, resource contention.



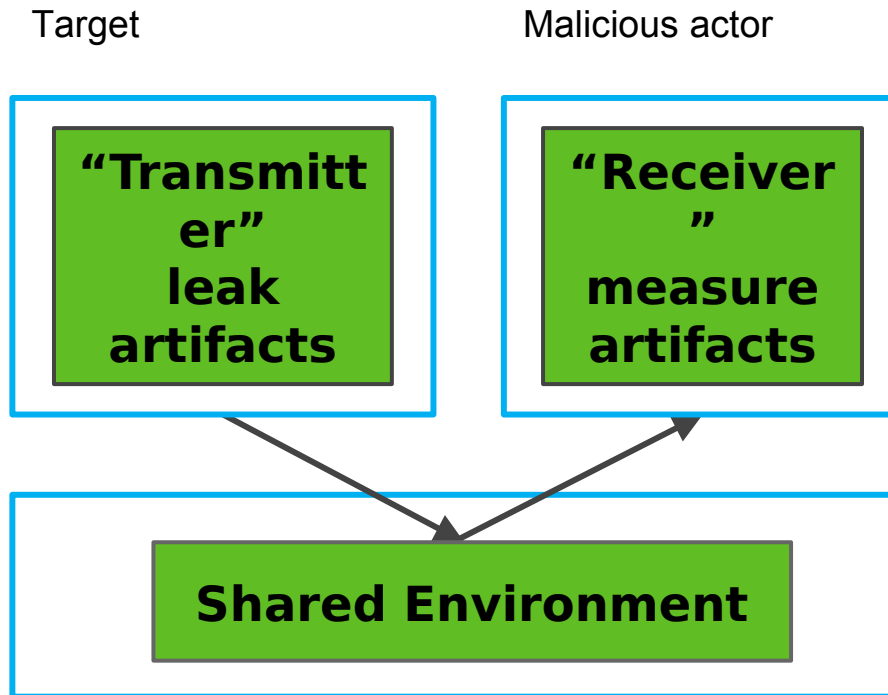
Information gained through recordable changes in the system



Powered sampled at even intervals across time.

Side Channel Checklist

- Transmitter
 - Deterministic cause and effect.
- Receiver
 - Record changes in environment without altering its readings
- Medium
 - Shared environment
 - Accountable sources of noise

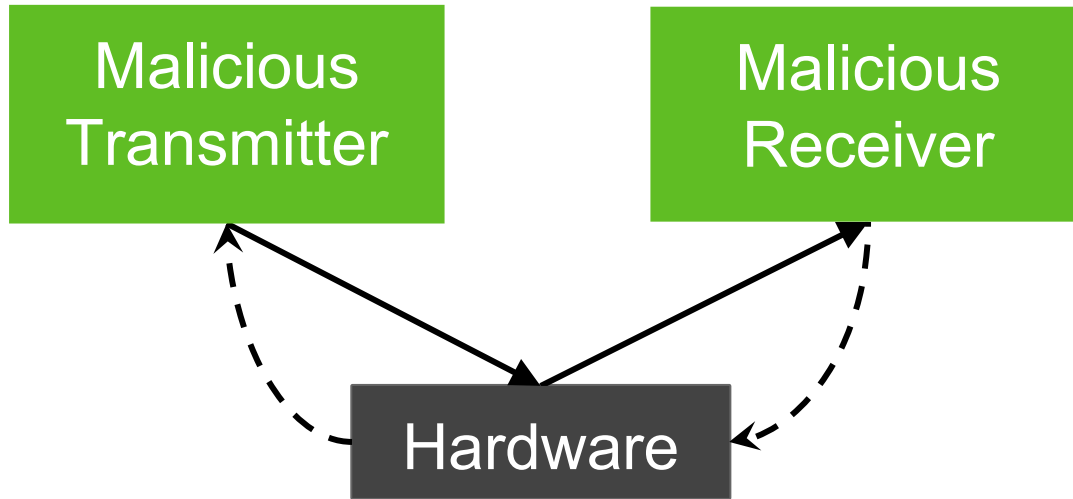


Targeting Hardware

The Hidden Attack Surface

TRAIL
OF
BITS

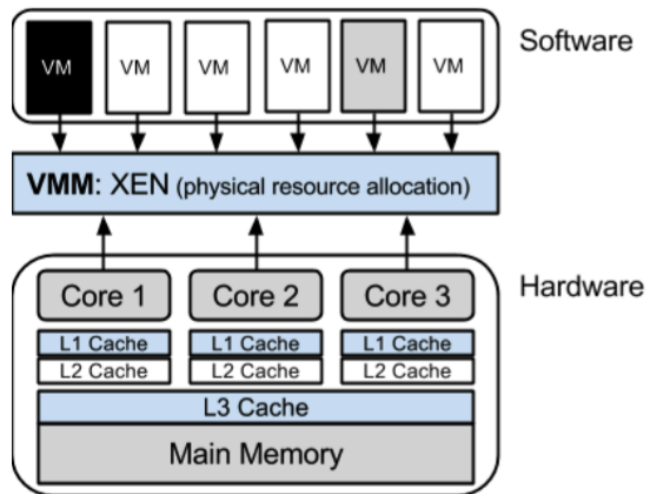
Communication Between Processes Using Hardware



Available Hardware

Shared environment on computers, accessible from software processes. Hardware resources shared between processes.

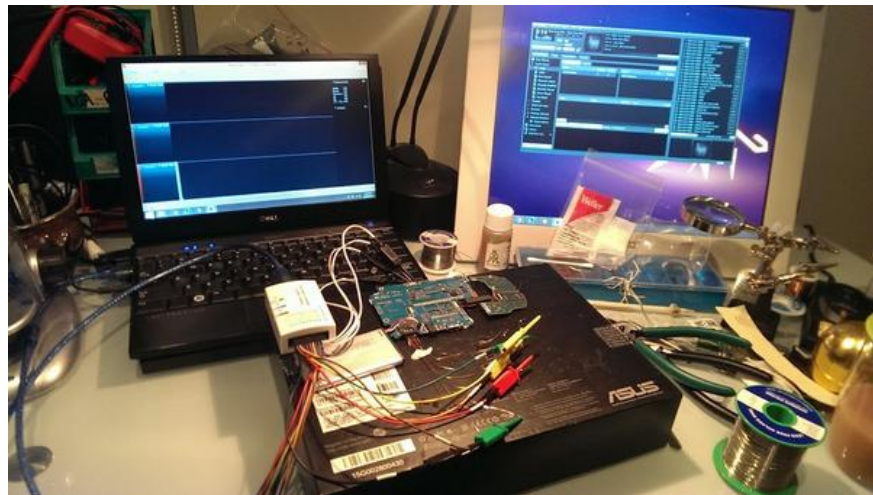
- Processors (CPU/ GPU)
- Cache Tiers
- System Buses
- Main Memory
- Hard Disk Drive



Side Channel Attacks Over Hardware

Physical co-location leads to side channel vulnerabilities.

- Processes share hardware resources
- Dynamic translation based on need
- Allocation causes contention



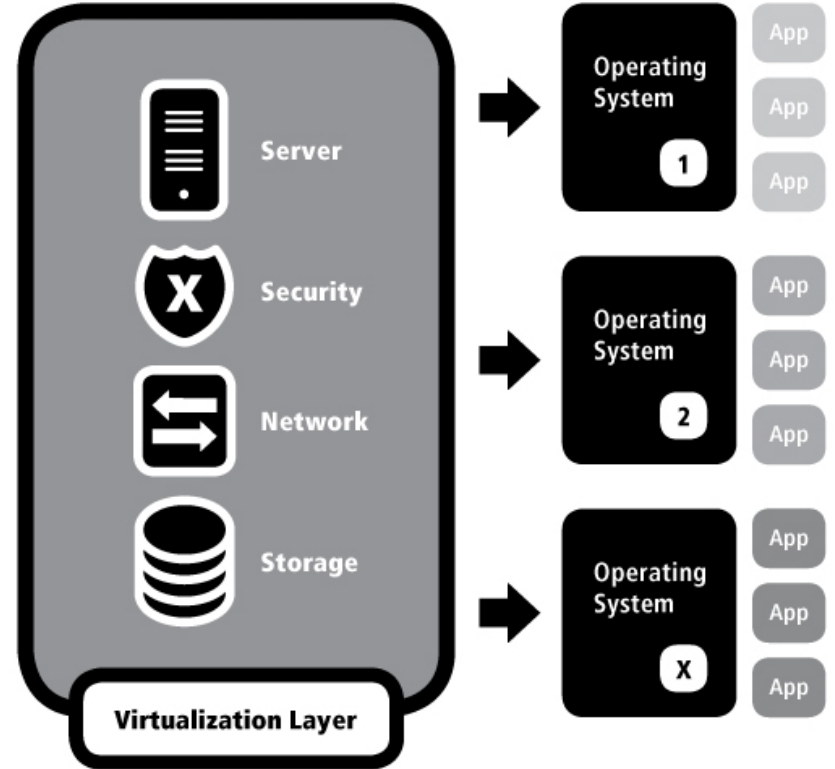
Cloud Computing (IaaS)

Perfect environment for hardware based side channels:

- Virtual instances
- Hypervisor schedules resources between all processors on a server

Dynamic allocation

- Reduces cost



Vulnerable Scenarios in the Cloud

- Sensitive data stored remotely
- Vulnerable host
- Untrusted host
- Co-located with a foreign VM



Building A Novel Attack

A Side Channel Recipe

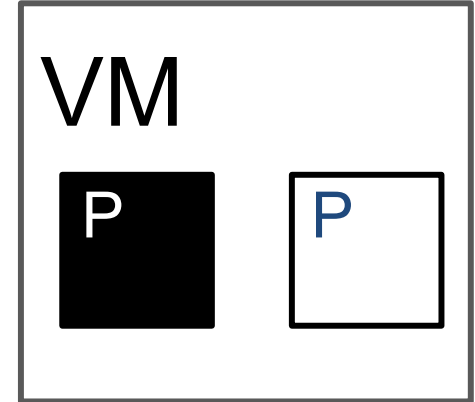
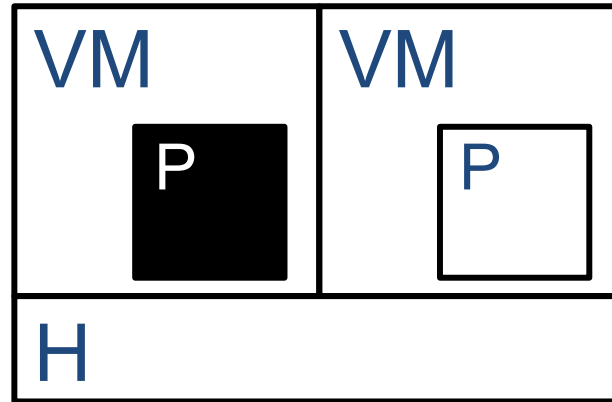
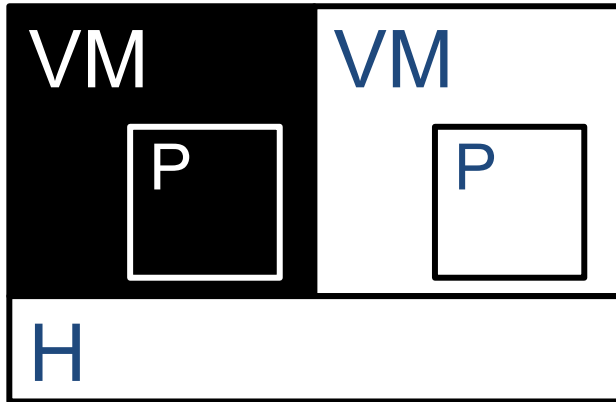
TRAIL
OF
BITS

Cloud Computing Side Channel Scenarios

Shared hardware

Dynamically allocated hardware resources

Co-Location with adversarial VMs, infected VMs, or Processes



Cloud Computing Side Channel - Primitives



Medium: Shared artifact from a hardware unit

"Privilege Separation": Virtual Machine or process

Method: Information gained through recordable changes in the system

Vulnerability: Translation between physical and virtual, dynamic!

First Ingredient: Hardware Medium

Choose Medium: Measure shared hardware unit's changes over time

- Cache
- Processor
- System Bus
- Main Memory
- HDD



Second Ingredient: Measuring Device

Choose Vulnerability: Measure artifact of shared resource.

- Timing attacks (usually best choice)
 - Cache misses, stored value farther away in memory
- Value Errors
 - Computation returns unexpected result
- Resource contention
 - Locking the memory bus
- Other measurements recordable from inside a process, in a VM



Third Ingredient: Attack Model

Choose S/R Model: What processes are involved in creating the channel depend on intended use cases.

- Transmit only
 - Application: DoS Attack
- Sender only
- Record only
 - Application: Crypto key theft
- Receiver only
- Bi-way
 - Application: Communication channel



Some channels are easier than others....



Case Study 1: Locking the memory bus

- Pro: efficient, no noise, good bandwidth
- Con: highly noticeable

Case Study 2: Everyone loves Cache.

- Pro: hardware medium is 'static'
- Con: most common, mitigations are quickly developed

Some channels are easier than others....



Technical Difficulties:

- Querying the specific hardware unit
- Difficulty/ reliability unique to each hardware unit
- Number of repeated measurements possible
- Frequency of measurements allowed

Measuring Devices for Hardware Mediums



Hardware Medium	Transmitting Mechanism	Reception Mechanism
Processor	Processor Register and Functional Unit Resources Contention	Time Compared Against Threshold
Cache Tier	Prime-Probe, Shared Cache Functionality	Time Compared Against Threshold
System Bus	System Bus Restricted Access Contention	Measurement of Memory Access Capabilities
Main Memory	Prime-Probe, Shared Main Memory Storage	Measurement of Memory Access Capabilities
Hard Disk Drive	Prime-Probe, Shared Disk Drive Data Access	Time Compared Against Threshold

Some Example Hardware Side Channels



Medium	Transmission	Reception	Constraints
L1 Cache	Prime Probe	Timing	Need to Share Processor Space
L2 Cache	Prime Probe/ Preemption	Timing	Caches Missing Causes Noise
Main Memory	SMT Paging	Measure Address Space	Peripheral Threads Create Noise
Memory Bus	Lock & Unlock Memory Bus	Measure Access	Halts all Processes Requiring the Bus
CPU Functional Units	Resource Eviction & Usage	Timing	mo' Threads, mo' Problems
Hard drive	Hard Disc Contention - Access Files Frantically	Timing	Dependent on multiple readings of files

A Novel Attack

- 1) **Medium:**
CPU Pipeline Optimization
- 2) **Vulnerability:**
Erroneous Values. Computation returns unexpected result (SMT optimizations).
- 3) **Model:**
Develop both a sender and receiver

General setup: Cross VM or Process.

CPU Optimizations

Uses of Out-Of-Order Execution

TRAIL
OF
BITS

A Novel Attack



Side Channel exploiting the pipeline's common optimization of re-ordering instructions.

- Regardless of process ownership
- Some re-ordering fails and computation result changes

Receiver: Measuring OoOE



8.2.3.4 Loads May Be Reordered with Earlier Stores to Different Locations

The Intel-64 memory-ordering model allows a load to be reordered with an earlier store to a different location. However, loads are not reordered with stores to the same location.

The fact that a load may be reordered with an earlier store to a different location is illustrated by the following example:

Example 8-3. Loads May be Reordered with Older Stores

Processor 0	Processor 1
<code>mov [_x], 1</code> <code>mov r1, [_y]</code>	<code>mov [_y], 1</code> <code>mov r2, [_x]</code>
Initially $x = y = 0$ $r1 = 0$ and $r2 = 0$ is allowed	

Synched

THREAD 1

store [X], 1

load r1, [Y]

store [X], 1

load r1, [Y]

load r1, [Y]

store [X], 1

THREAD 2

store [Y], 1

load r2, [X]

store [Y], 1

load r2, [X]

load r2, [X]

store [Y], 1

Asynchronous

Out of Order Execution

r1 = r2 = 1

r1 = 0 r2 = 1

r1 = r2 = 0

```

push    edi
call   sub_314623
test   eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], ebx
jnz   short loc_313066
mov    eax, [ebp+var_70]
cmp    eax, [ebp+var_84]
jb     short loc_313066
sub    eax, [ebp+var_84]
ret

```

```

push    esi
lea   eax, [ebp+arg_0]
shl   eax, 1D0h
mov    esi, 1D0h
push  esi
push  [ebp+arg_4]

```

```

call   sub_314623
eax, eax
test   eax, eax
cmp    [ebp+arg_0], esi
jz     short loc_31306F
loc_313066:
push  1
call   sub_31411B

```

```

call   sub_3140F3
test   eax, eax
cmp    [ebp+arg_0], esi
call   sub_314013
jmp    short loc_31308C

```

```

loc_31307D:
and    eax, 0FFFFFFh
or     eax, 80070000h

```

```

loc_31308C:
mov    [ebp+var_4], eax

```

Receiver: Measuring OoOE

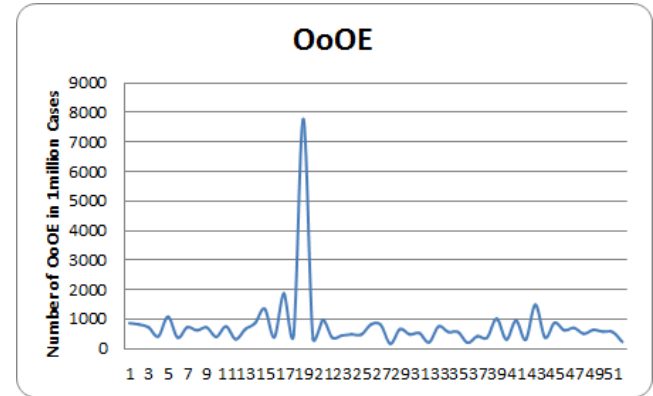
```

int X,Y,count_OoOE;
...initialize semaphores Sema1 & Sema2...
pthread_t thread1, thread2;
pthread_create(&threadN, NULL, threadNFunc, NULL);

for (int iterations = 1; ; iterations++)
    X,Y = 0;
    sem_post(beginSema1 & beginSema2);
    sem_wait(endSema1 & endSema2);

if (r1 == 0 && r2 == 0)
    count_OoOE ++;

```



Sender: Transmit OoOE

Force Deterministic Memory Reordering:

- Compile-time vs Runtime Reordering

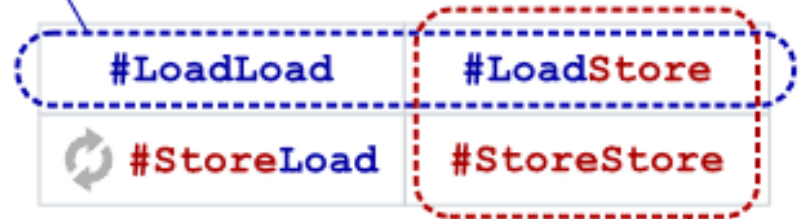
Runtime:

- Usually strong memory model: x86/64 (mostly sequentially consistent)
- Weaker models (data dependency re-ordering): arm, powerpc

Barriers:

- 4 types of run time reordering barriers

acquire semantics



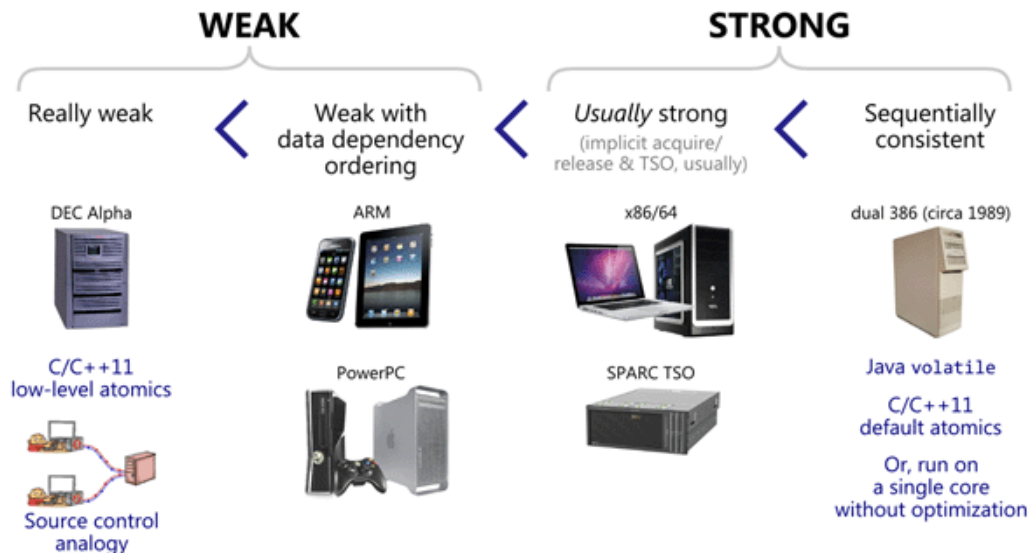
release semantics

Sender: Transmit OoOE

Memory Fences

Mfence:

- x86 instruction full memory barrier
- prevents memory reordering of any kind
- order of 100 cycles per operation
- lock-free programming on SMT multiprocessors



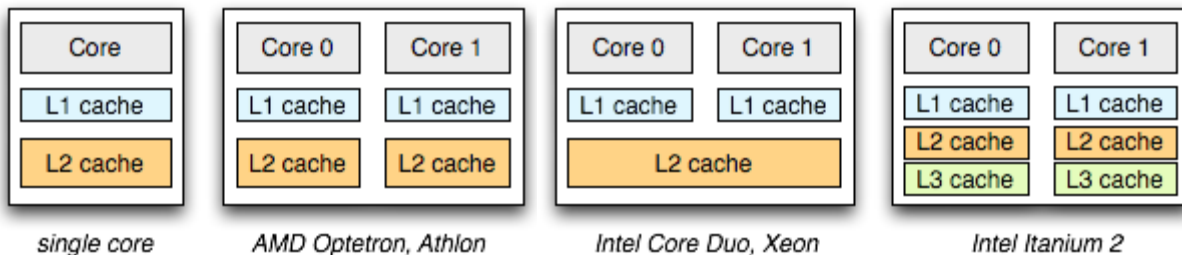
Sender: Transmit OoOE

`mfence` (x86)

#StoreLoad unique prevents r1=r2=0



Testing: Hardware Architectures



Lab Setup:

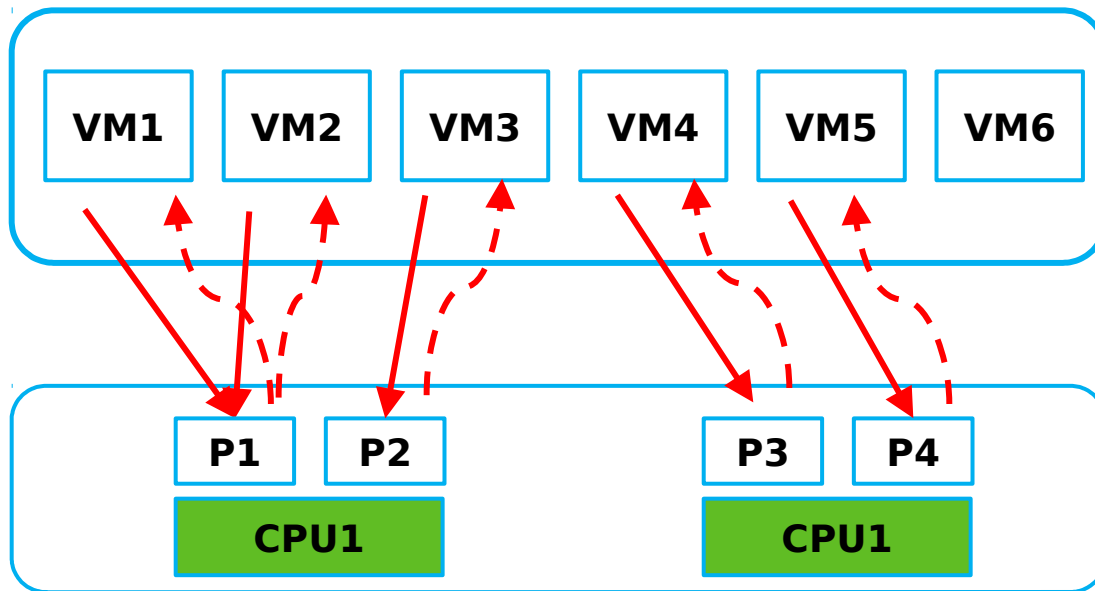
- Intel's Core Duo, Xeon Architecture
- Each processor has two cores
- The Xen hypervisor schedules between all processors on a server
- Each core then allocates processes on its pipeline

Notes:

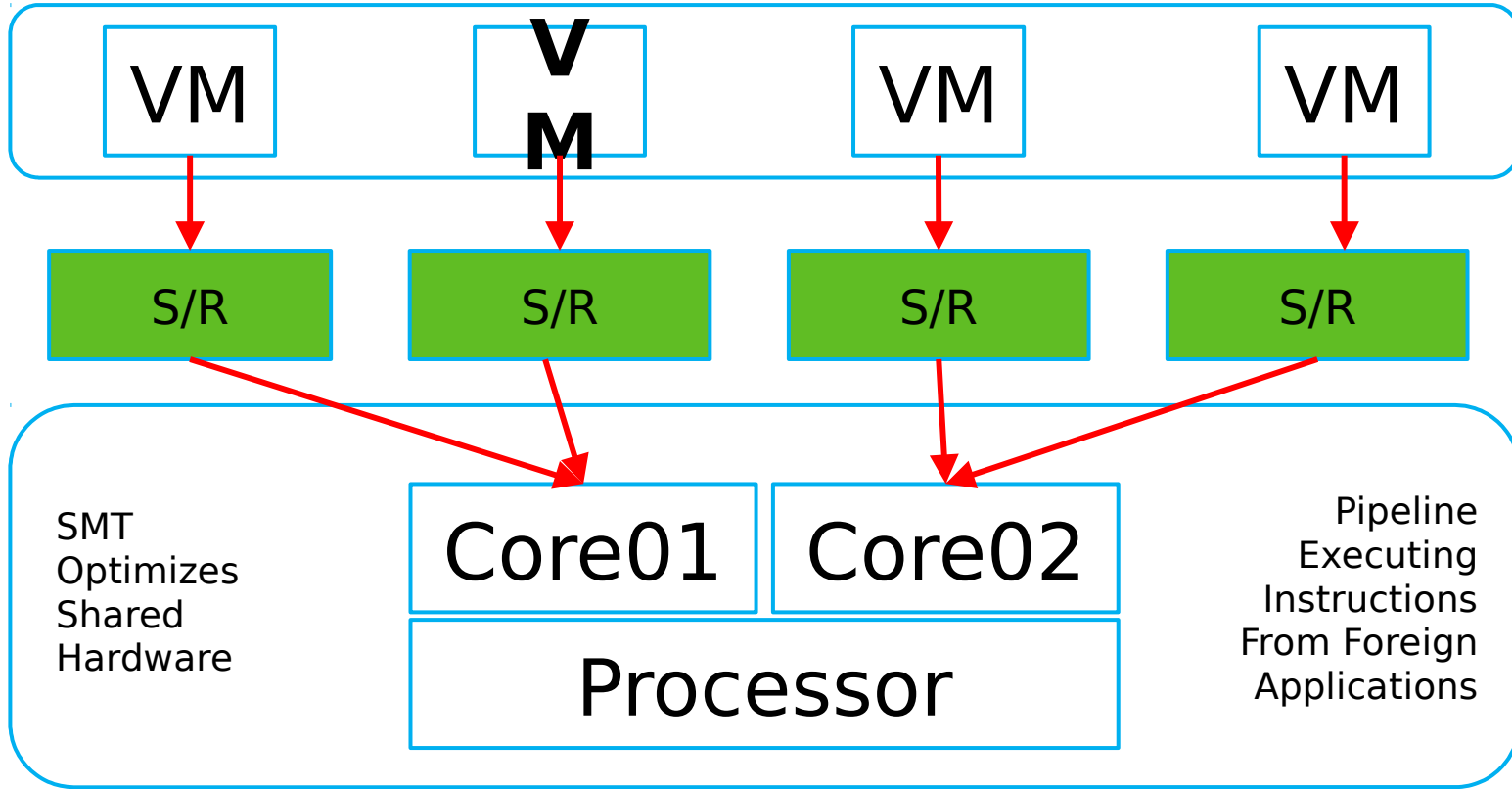
- Multiple processes run on a single pipeline (SMT)
- Relaxed memory model

Testing: Setup

6 Windows 7 VM's

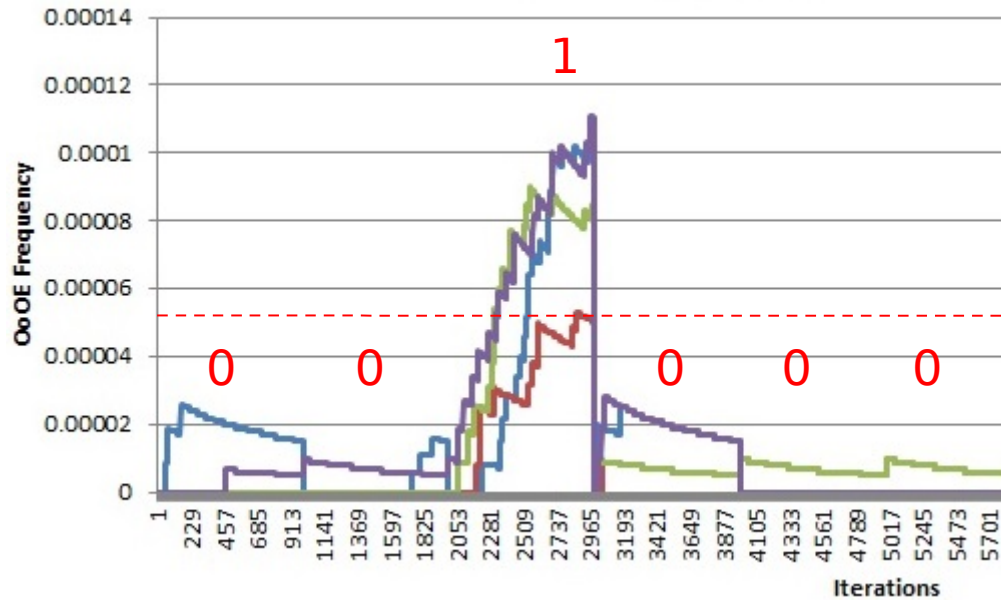


Testing: Setup



Testing: Results

Sending signal: 001000.



Testing: Results



Benefits:

- Harder for an intelligent hypervisor to detect, quiet
- Eavesdropping sufficiently mutilates channel
- System artifacts sent and queried dynamically
- Not affected by cache misses
- Channel amplified with system noise
- Immediately useful for malware, leaking system behavior, environmental keying, algorithm identification

More Info: <https://www.sophia.re/SC>

Defenses

TRAIL
OF
BITS

Defensive Mechanisms: Hardware

Protected Resource Ownership:

- Isolating VM's
- Turn off hyperthreading
- Blacklisting resources for concurrent threads
- Downside: removes optimizations or benefits of the cloud



Defensive Mechanisms: Hypervisor



Anomaly detection:

- Specification
- Pattern recognition
- Records average OoOE patterns
- Predicts what to expect

Defensive Mechanisms: Software

Control Flow Changes:

- Hardening software with *Noise*
- Force specific execution patterns (i.e. constant time loops, ...)
- Avoid using certain resources
- Downside: compiler, hardware optimizations lost



Virtualization Considerations



Side Channel Potential:

- More resource sharing
- More dynamic optimizations
- Virtualization more popular
- Malware

Things to Consider:

- Cloud Side Channels apply to anything with virtualization (i.e. VM's)
- Hypervisors are easy targets: Vulnerable host

i.e. "Xenpwn", paravirtualized driver attack: INFILTRATECon 2016

The Future

TRAIL
OF
BITS

Optimizations!



Intel Architecture Optimization Manual

Order Number 242816-003

1997

Optimizations!

2.2 THE PENTIUM[®] PRO PROCESSOR

The Pentium Pro processor family uses a dynamic execution architecture that blends out-of-order and speculative execution with hardware register renaming and branch prediction. These processors feature an in-order issue pipeline, which breaks IA processor macroinstructions into simple, micro-operations called micro-ops or μ ops, and an out-of-order, superscalar processor core, which executes the micro-ops. The out-of-order core of the processor contains several pipelines to which integer, branch, floating-point and memory execution units are attached. Several different execution units may be clustered on the same pipeline. For example, an integer arithmetic logic unit and the floating-point execution units (adder, multiplier and divider) share a pipeline. The data cache is pseudo-dual ported via interleaving, with one port dedicated to loads and the other to stores. Most simple operations (such as integer ALU, floating-point add and floating-point multiply) can be pipelined with a throughput of one or two operations per clock cycle. The floating-point divider is not pipelined. Long latency operations can proceed in parallel with short latency operations.

The Pentium Pro processor pipeline contains three parts: (1) the in-order issue front-end, (2) the out-of-order core, and (3) the in-order retirement unit. Figure 2-3 details the entire Pentium Pro processor pipeline.

Optimizations!

- Processor Register and Functional Unit
- Out-Of-Order Execution
- Speculative Execution
- Branch Prediction
- 1 Pipeline, multiple execution units
 - i.e. Integer ALU and FPU (adder, multiplier and divider) share a pipeline
- Data cache pseudo-dual ported via interleaving
- “ Long latency operations can proceed in parallel with short latency operations.”

Optimizations!

- Processor Register and Functional Unit
- Out-Of-Order Execution
- Speculative Execution
- Branch Prediction
- 1 Pipeline, multiple execution units
 - i.e. Integer ALU and FPU (adder, multiplier and divider) share a pipeline
- Data cache pseudo-dual ported via interleaving
- “ Long latency operations can proceed in parallel with short latency operations.”

Optimizations!



L1 Data Cache Loads can:

- Read data before preceding stores when the load address and store address ranges are known not to Conflict.
- Be carried out speculatively, before preceding branches are resolved.
- Take cache misses out of order and in an overlapped manner.

Optimizations!



Speculative Execution

- **Bad speculation** - The pipeline performs speculative execution of instructions that never successfully retire. The most common case is a branch misprediction where the pipeline predicts a branch target in order to keep the pipeline full instead of waiting for the branch to execute. If the processor prediction is incorrect it has to flush the pipeline without retiring the speculated instructions.

Speculative Execution



```
mov rax,[addr_0]
```

```
mov rbx,[addr_1]
```

Speculative Execution



```
mov rax,[addr_0]
```

```
add rax, 1
```

```
mov rbx,[rax + addr_1]
```

Speculative Execution



```
mov rax,[addr_0]
```

```
add rax, 1
```

```
mov rbx,[rax + addr_1] ← Time Memory Load
```

Speculative Execution

Uses: Arbitrary Kernel Memory Leak!




```
mov rax,[k_addr]
```

- Interrupt occurs
- Undefined behavior
 - Timing of finished instruction execution and actual retirement
 - mov potentially sets the results in the reorder buffer

Goal: Speculatively execute instructions after mov, based on reorder buffer value.

Speculative Execution



<code>syscall</code>		Force target into cache
<code>mov rax,[k_addr]</code>		Guessed address
<code>add rax, 1</code>		
<code>mov rbx,[rax + addr_1]</code>		Time to validate guess

Speculative Execution: Tomasulo algorithm



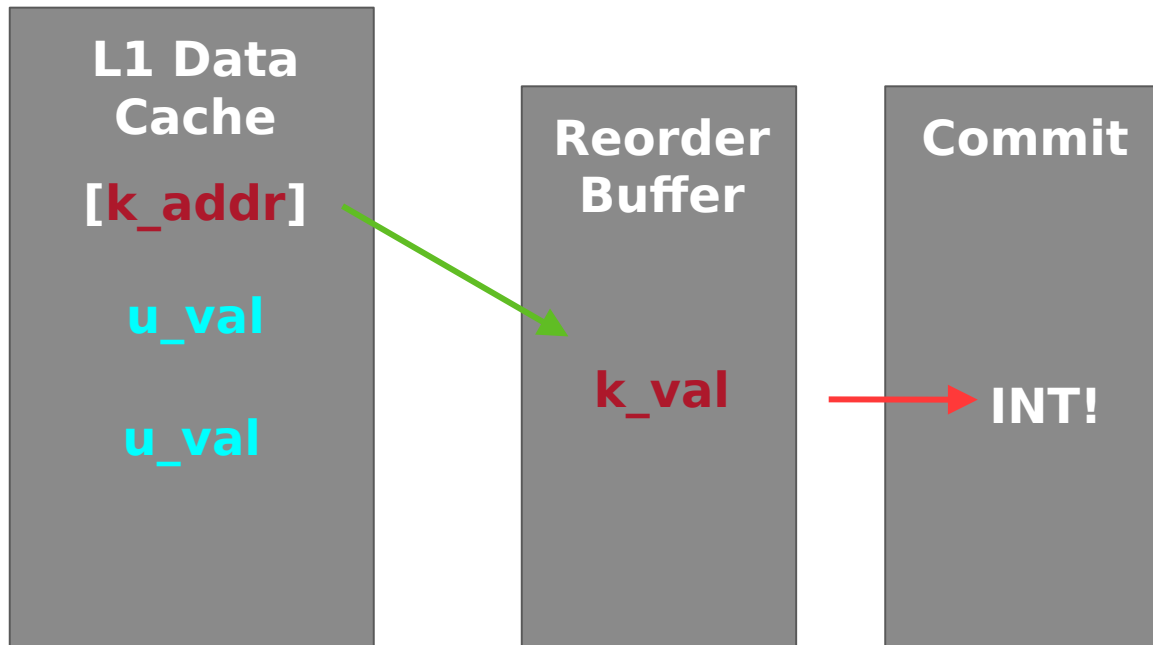
syscall

mov rax,[k_addr]

add rax, 1

mov rbx,[rax + addr_1]

Speculative Execution: Tomasulo algorithm



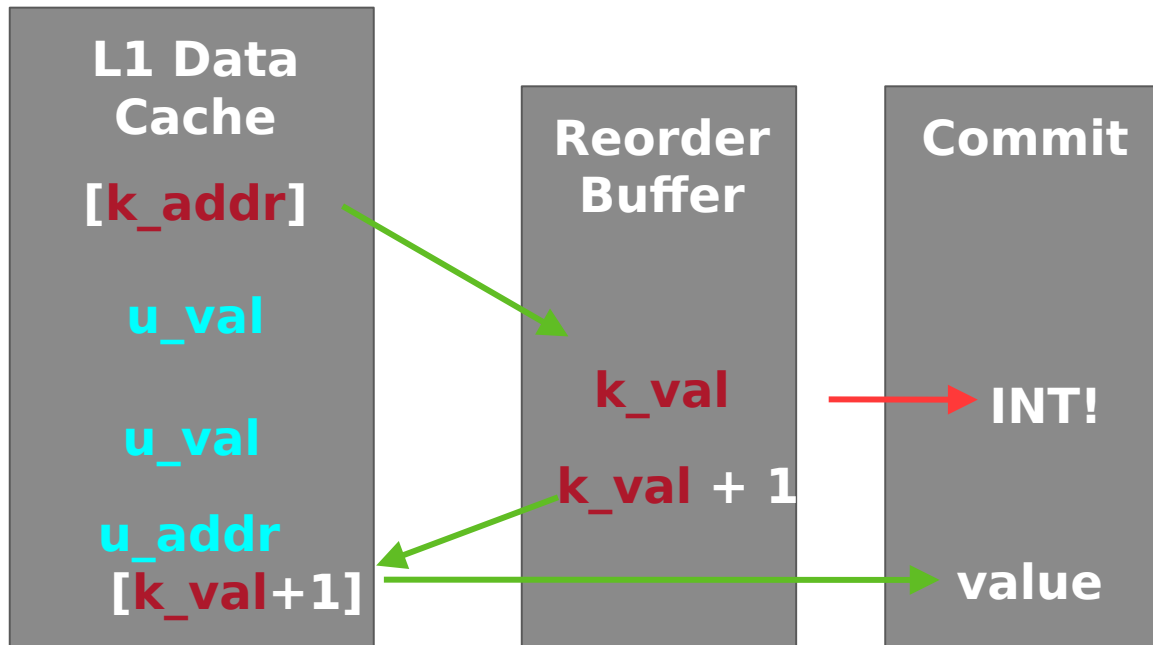
syscall

```
mov rax,[k_addr]
```

```
add rax, 1
```

```
mov rbx,[rax + addr_1]
```

Speculative Execution: Tomasulo algorithm



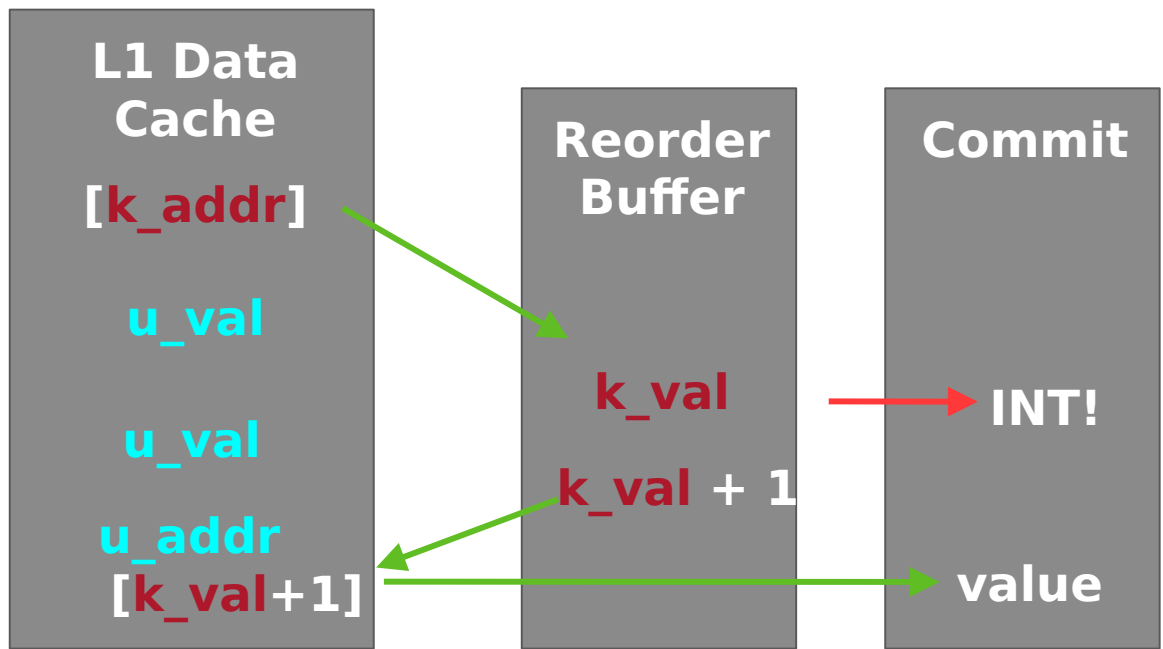
syscall

```
mov rax,[k_addr]
```

```
add rax, 1
```

```
mov rbx,[rax + addr_1]
```

Speculative Execution: Tomasulo algorithm



time!

```
syscall
mov rax,[k_addr]
add rax, 1
mov rbx,[rax + addr_1]
```

Speculative Execution



Test target: Intel Broadwell CPU

- While goal `k_addr` value might not be given directly
- Use cache side channel to verify result or not
- Failed on this target, but...
 - Does process illegal read from `k_addr` (!)
 - Does not copy value into reorder buffer :<
 - Loads from data cache during speculative execution
 - Speculative execution & data loads do occur after violation of kernel/user read

Speculative Execution

Test target: Intel Broadwell CPU

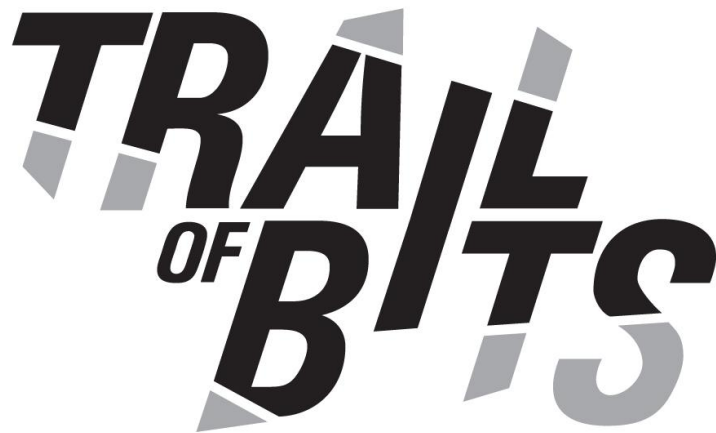
- While goal k_addr value might not be given directly
- Use cache side channel to verify result or not
- Failed on this target, but...
 - Does process illegal read from k_addr (!)
 - Does not copy value into reorder buffer :<
 - Loads from data cache during speculative execution
 - Speculative execution & data loads do occur after violation of kernel/user read

To be continued....

Acknowledgements

co-author: Jeremy Blackthorne
advisor: Bulent Yener

Trail of Bits
Ryan Stortz, Jeff Preshing, Anders Fog



<https://www.sophia.re/SC>

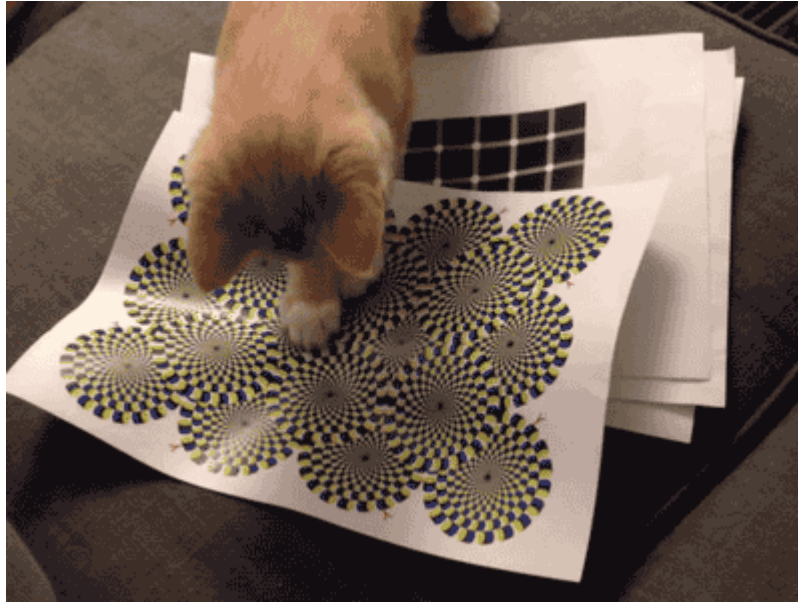
<http://preshing.com/20120515/memory-reordering-caught-in-the-act/>

<http://blog.stuffedcow.net/2014/01/x86-memory-disambiguation/>

<https://cyber.wtf/2017/07/28/negative-result-reading-kernel-memory-from-user-mode/>

<https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf>

Any Questions?



IRC: quend

email: sophia@trailofbits.com

website: www.sophia.re

The Bad Neighbor

Out-of-Order Execution and Its Applications

Sophia d'Antoine
November 7th, 2017

TRAIL
OF
BITS

whoami

Masters in CS from RPI

- Exploiting Intel's CPU pipelines

Work at Trail of Bits

- Senior Security Researcher
- Program Analysis / Ethereum Smart Contracts

DEFCON (CTF), CSAW

Stats

- 12 Conferences Worldwide
- 3 Program Committees
- 2 Security Panels
- 1 Paper Published
- 1 Keynote



Blackhat, HITB, RECon, Cansecwest. Mention CTF got into it. DEFCON CTF
Program analysis work automation, etc. Cyber Transition Team?

Side Channels

Hardware Side Channels in Virtualized Environments

TRAIL
OF
BITS

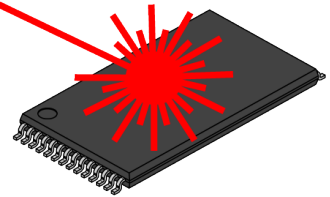
What are side channel attacks?

- Attacker can observe the target system. Must be 'neighboring' or co-located.
- Ability to repeatedly query the system for leaked artifacts.
- Artifacts: changes in how a process interacts with the computer

Variety of Side Channels

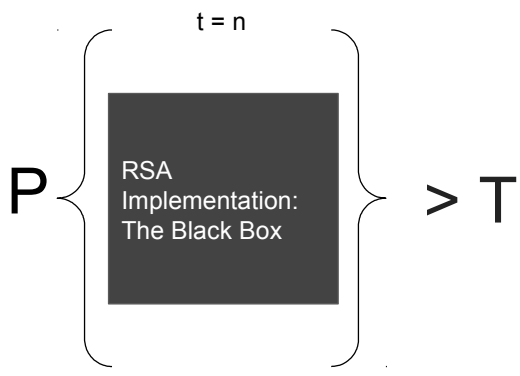
Different target systems implies different methods for observing.

- Fault attacks
 - Requires access to the hardware.
- Simple power analysis
 - Requires proximity to the system.
 - Power consumption measurement mapped to behavior.
- Different power analysis
 - Requires proximity to the system.
 - Statistics and error correction gathered over time.
- Timing attacks
 - Requires same process co-location.
 - Network packet delivery, cache misses, resource contention.



Crypto stuff

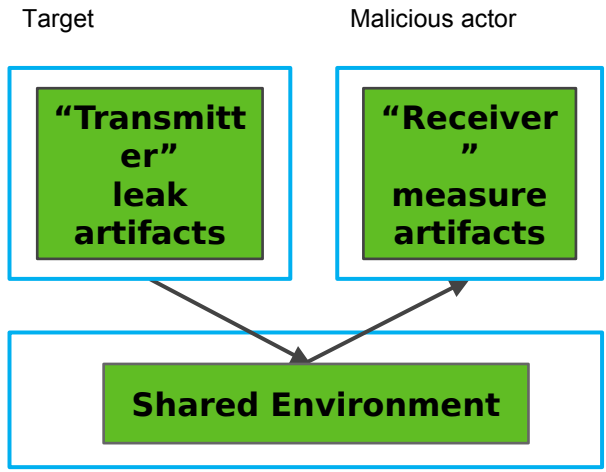
Information gained through recordable changes in the system



Powered sampled at even intervals across time.

Side Channel Checklist

- Transmitter
 - Deterministic cause and effect.
- Receiver
 - Record changes in environment without altering its readings
- Medium
 - Shared environment
 - Accountable sources of noise



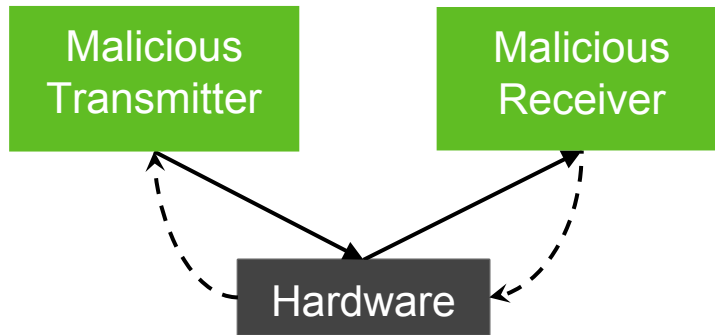
Software AND Hardware both

Targeting Hardware

The Hidden Attack Surface

TRAIL
OF
BITS

Communication Between Processes Using Hardware

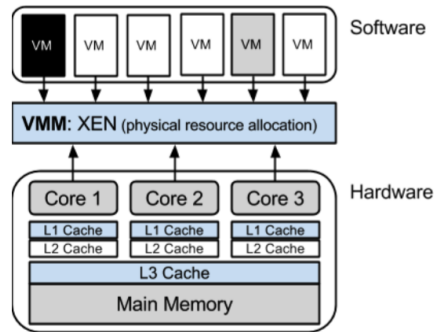


Software AND Hardware both

Available Hardware

Shared environment on computers, accessible from software processes. Hardware resources shared between processes.

- Processors (CPU/ GPU)
- Cache Tiers
- System Buses
- Main Memory
- Hard Disk Drive



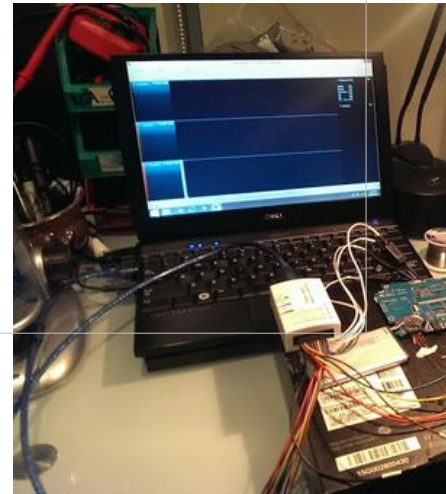
Software AND Hardware both

Side Channel Attacks Over Hardware

Physical co-location leads to side channel vulnerabilities.

- Processes share hardware resources
- Dynamic translation based on need
- Allocation causes contention

Software AND Hardware both



Cloud Computing (IaaS)

Perfect environment for hardware based side channels:

- Virtual instances
- Hypervisor schedules resources between all processors on a server

Dynamic allocation

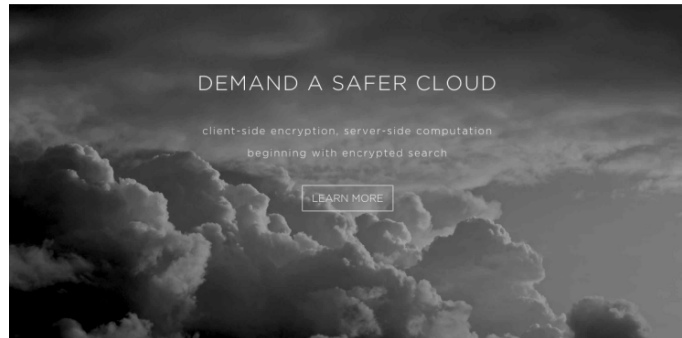
- Reduces cost

Software AND Hardware both



Vulnerable Scenarios in the Cloud

- Sensitive data stored remotely
- Vulnerable host
- Untrusted host
- Co-located with a foreign VM



Software AND Hardware both

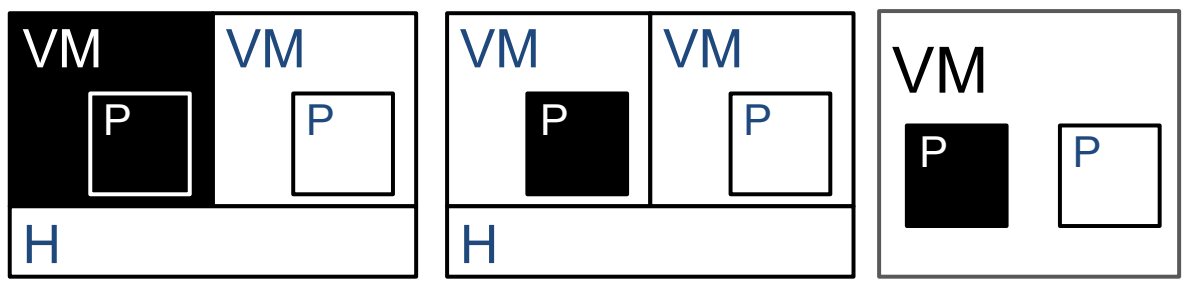
Building A Novel Attack

A Side Channel Recipe

TRAIL
OF
BITS

Cloud Computing Side Channel Scenarios

- Shared hardware
- Dynamically allocated hardware resources
- Co-Location with adversarial VMs, infected VMs, or Processes



Software AND Hardware both

Cloud Computing Side Channel - Primitives

Medium: Shared artifact from a hardware unit

"Privilege Separation": Virtual Machine or process

Method: Information gained through recordable changes in the system

Vulnerability: Translation between physical and virtual, dynamic!

Software AND Hardware both

First Ingredient: Hardware Medium

Choose Medium: Measure shared hardware unit's changes over time

- Cache
- Processor
- System Bus
- Main Memory
- HDD



Software AND Hardware both

Second Ingredient: Measuring Device

Choose Vulnerability: Measure artifact of shared resource.

- Timing attacks (usually best choice)
 - Cache misses, stored value farther away in memory
- Value Errors
 - Computation returns unexpected result
- Resource contention
 - Locking the memory bus
- Other measurements recordable from inside a process, in a VM



Software AND Hardware both

Third Ingredient: Attack Model

Choose S/R Model: What processes are involved in creating the channel depend on intended use cases.

- Transmit only
 - Application: DoS Attack
 - Sender only
- Record only
 - Application: Crypto key theft
 - Receiver only
- Bi-way
 - Application: Communication channel

Software AND Hardware both



Some channels are easier than others....

Case Study 1: Locking the memory bus

- Pro: efficient, no noise, good bandwidth
- Con: highly noticeable

Case Study 2: Everyone loves Cache.

- Pro: hardware medium is 'static'
- Con: most common, mitigations are quickly developed

Software AND Hardware both

Some channels are easier than others....

Technical Difficulties:

- Querying the specific hardware unit
- Difficulty/ reliability unique to each hardware unit
- Number of repeated measurements possible
- Frequency of measurements allowed

Software AND Hardware both

Measuring Devices for Hardware

TRAJ
BITS

M_H

Hardware Medium	Transmitting Mechanism	Reception
Processor	Processor Register and Functional Unit Resources Contention	Time Comp Threshold
Cache Tier	Prime-Probe, Shared Cache Functionality	Time Comp Threshold
System Bus	System Bus Restricted Access Contention	Measureme Access Cap
Main Memory	Prime-Probe, Shared Main Memory Storage	Measureme Access Cap
Hard Disk Drive	Prime-Probe, Shared Disk Drive Data Access	Time Comp Threshold

Software AND Hardware Both

Some Example Hardware Side Channels



Medium	Transmission	Reception	Constraints
L1 Cache	Prime Probe	Timing	Need to Share Processor Space
L2 Cache	Prime Probe/ Preemption	Timing	Caches Missing Causes Noise
Main Memory	SMT Paging	Measure Address Space	Peripheral Threads Create Noise
Memory Bus	Lock & Unlock Memory Bus	Measure Access	Halts all Processes Requiring the Bus
CPU Functional Units	Resource Eviction & Usage	Timing	mo' Threads, mo' Problems
Hard drive	Hard Disc Contention - Access Files Frantically	Timing	Dependent on multiple readings of files

See how they all follow the recipe. Existing work. Abstraction is KEY!

A Novel Attack

- 1) **Medium:**
CPU Pipeline Optimization
- 2) **Vulnerability:**
Erroneous Values. Computation returns unexpected result (SMT optimizations).
- 3) **Model:**
Develop both a sender and receiver

General setup: Cross VM or Process.

Smt optimizations are key for this pipeline attack. Optimizations are a great vulnerability in general.

CPU Optimizations

Uses of Out-Of-Order Execution

TRAIL
OF
BITS

A Novel Attack

Side Channel exploiting the pipeline's common optimization of re-ordering instructions.

- Regardless of process ownership
- Some re-ordering fails and computation result changes

Smt optimizations are key for this pipeline attack. Optimizations are a great vulnerability in general.

Receiver: Measuring OoOE



8.2.3.4 Loads May Be Reordered with Earlier Stores to Different Locations

The Intel-64 memory-ordering model allows a load to be reordered with an earlier store to a different location. However, loads are not reordered with stores to the same location.

The fact that a load may be reordered with an earlier store to a different location is illustrated by the following example:

Example 8-3. Loads May be Reordered with Older Stores

Processor 0	Processor 1
<code>mov [_x], 1</code>	<code>mov [_y], 1</code>
<code>mov r1, [_y]</code>	<code>mov r2, [_x]</code>
Initially $x = y = 0$ $r1 = 0$ and $r2 = 0$ is allowed	

Smt optimizations are key for this pipeline attack. Optimizations are a great vulnerability in general.

Synched

THREAD 1

store [X], 1
load r1, [Y]

THREAD 2

store [Y], 1
load r2, [X]

=> r1 = r2 = 1

Asynched

[Y] store [X], 1
load r1, [Y]

[X] store [Y], 1
load r2, [X]

=> r1 = 0 r2 = 1

Out of Order Execution

[Y] load r1, [Y]
store [X], 1

[X] load r2, [X]
store [Y], 1

=> r1 = r2 = 0

```

push    edi
call    sub_314623
test    eax, eax
jz      short loc_31306D
cmp     [ebp+arg_0], ebx
short  loc_313066
mov     eax, [ebp+var_70]
cmp     eax, [ebp+var_84]
jb      short loc_313066
sub     eax, [ebp+var_84]
ret

loc_313066:
mov     esi, [ebp+arg_0], ebx
call    sub_314623
mov     esi, 140h
push   esi
push   [ebp+arg_4]
call    sub_314623
mov     eax, [ebp+arg_0]
call    sub_314623
short  loc_313066
loc_313066:
push   1
call    sub_31411B
loc_31306D:
call    sub_3140F3
mov     eax, eax
short  loc_31306C
loc_31307B:
sub_3140F3
and    eax, 00000000h
or     eax, 00070000h
loc_31308C:
mov     [ebp+var_41], eax

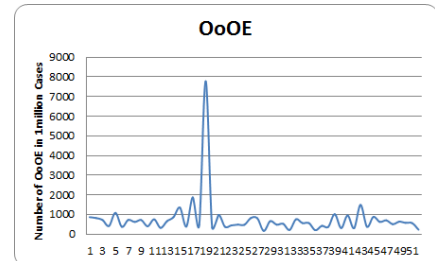
```

Receiver: Measuring OoOE

```
int X,Y,count_OoOE;
....initialize semaphores Sema1 & Sema2...
pthread_t thread1, thread2;
pthread_create(&threadN, NULL, threadNFunc, NULL);

for (int iterations = 1; ; iterations++)
    X,Y = 0;
    sem_post(beginSema1 & beginSema2);
    sem_wait(endSema1 & endSema2);

if (r1 == 0 && r2 == 0)
    count_OoOE ++;
```



Smt optimizations are key for this pipeline attack. Optimizations are a great vulnerability in general.

Sender: Transmit OoOE

Force Deterministic Memory Reordering:

- Compile-time vs Runtime Reordering

Runtime:

- Usually strong memory model: x86/64 (mostly sequentially consistent)
- Weaker models (data dependency re-ordering): arm, powerpc

Barriers:

- 4 types of run time reordering barriers



- `#StoreLoad` most expensive

Sender: Transmit OoOE



Memory Fences

Mfence:

- x86 instruction full memory barrier
- prevents memory reordering of any kind
- order of 100 cycles per operation
- lock-free programming on SMT multiprocessors



2 types of memory reordering, GCC
Multithreaded
Programs
Or pipeline type

Sender: Transmit OoOE

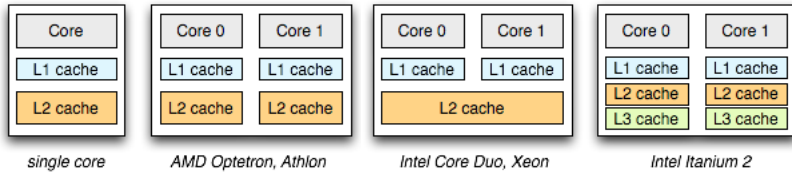
mfence (x86)

#StoreLoad unique prevents r1=r2=0



2 types of memory reordering, GCC
Multithreaded
Programs
Or pipeline type

Testing: Hardware Architectures



Lab Setup:

- Intel's Core Duo, Xeon Architecture
- Each processor has two cores
- The Xen hypervisor schedules between all processors on a server
- Each core then allocates processes on its pipeline

Notes:

- Multiple processes run on a single pipeline (SMT)
- Relaxed memory model

2 types of memory reordering, GCC

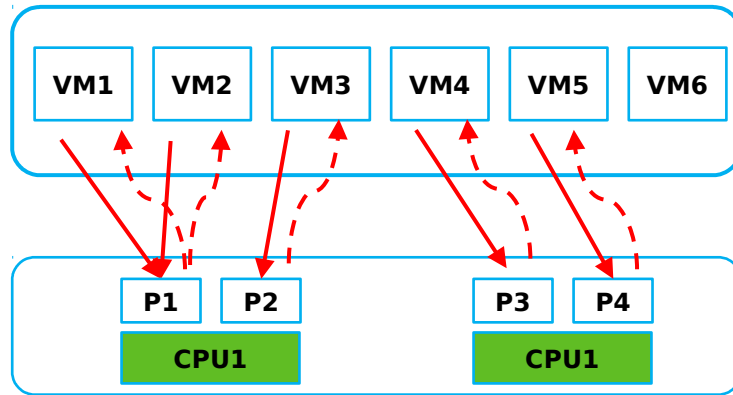
Multithreaded

Programs

Or pipeline type

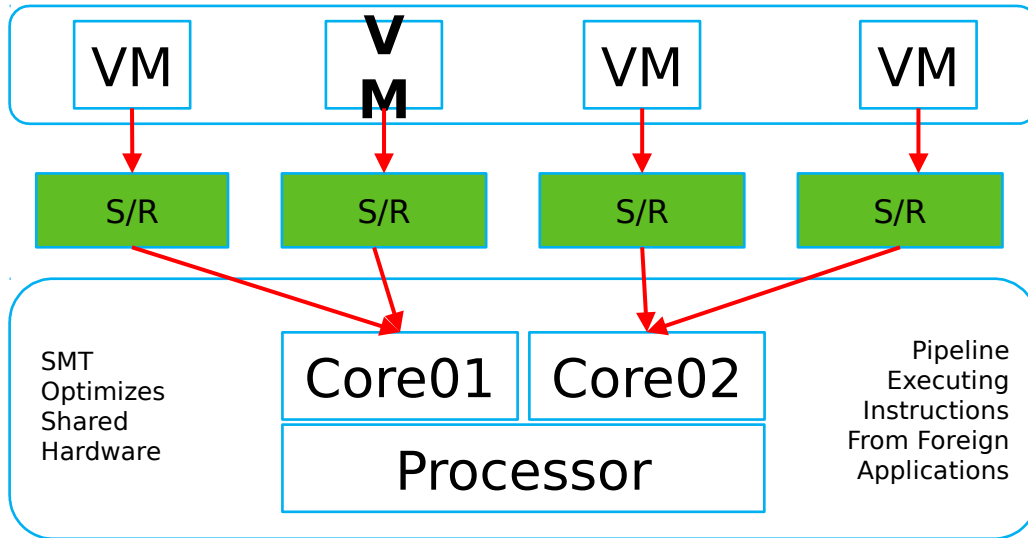
Testing: Setup

6 Windows 7 VM's



2 types of memory reordering, GCC
 Multithreaded
 Programs
 Or pipeline type

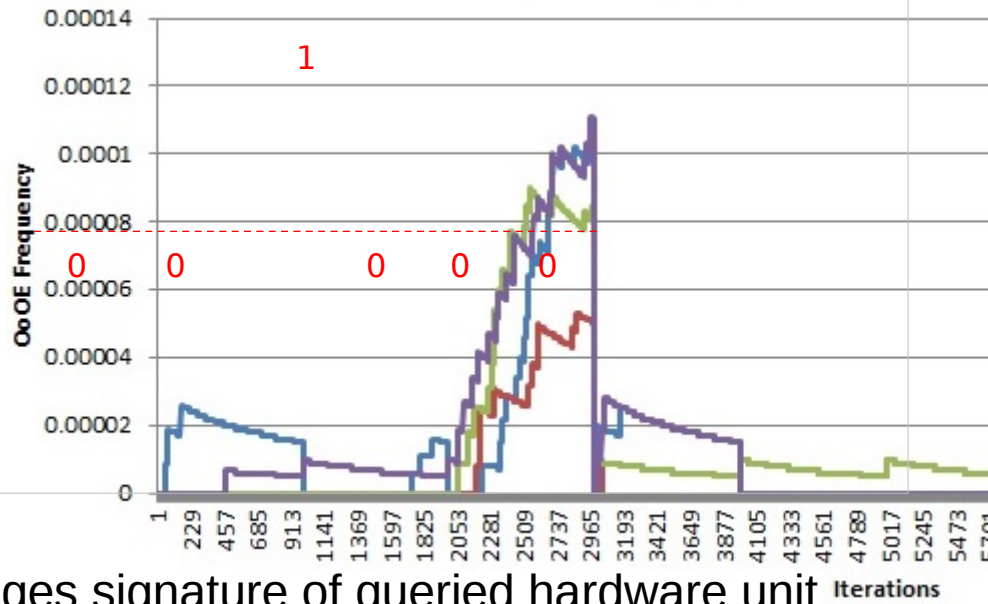
Testing: Setup



2 types of memory reordering, GCC
Multithreaded
Programs
Or pipeline type

Testing: Results

Sending signal: 001000.



Process changes signature of queried hardware unit over time

MALWARE USES ETC

Testing: Results

Benefits:

- Harder for a intelligent hypervisor to detect, quiet
- Eavesdropping sufficiently mutilates channel
- System artifacts sent and queried dynamically
- Not affected by cache misses
- Channel amplified with system noise
- Immediately useful for malware, leaking system behavior, environmental keying, algorithm identification

More Info: <https://www.sophia.re/SC>

Defenses

TRAIL
OF
BITS

Defensive Mechanisms: Hardware

Protected Resource Ownership:

- Isolating VM's
- Turn off hyperthreading
- Blacklisting resources for concurrent threads
- Downside: removes optimizations or benefits of the cloud



Defensive Mechanisms: Hypervisor



Anomaly detection:

- Specification
- Pattern recognition
- Records average OoOE patterns
- Predicts what to expect

Defensive Mechanisms: Software

Control Flow Changes:

- Hardening software with *Noise*
- Force specific execution patterns (i.e. constant time loops, ...)
- Avoid using certain resources
- Downside: compiler, hardware optimizations lost



Virtualization Considerations

Side Channel Potential:

- More resource sharing
- More dynamic optimizations
- Virtualization more popular
- Malware

Things to Consider:

- Cloud Side Channels apply to anything with virtualization (i.e. VM's)
- Hypervisors are easy targets: Vulnerable host

i.e. "Xenpwn", paravirtualized driver attack: INFILTRATECon 2016

The Future

TRAIL
OF
BITS

Optimizations!



Intel Architecture Optimization Manual

Order Number 242816-003

1997

Optimizations!

2.2 THE PENTIUM® PRO PROCESSOR

The Pentium Pro processor family uses a dynamic execution architecture that blends out-of-order and speculative execution with hardware register renaming and branch prediction. These processors feature an in-order issue pipeline, which breaks IA processor macroinstructions into simple, micro-operations called micro-ops or μ ops, and an out-of-order, superscalar processor core, which executes the micro-ops. The out-of-order core of the processor contains several pipelines to which integer, branch, floating-point and memory execution units are attached. Several different execution units may be clustered on the same pipeline. For example, an integer arithmetic logic unit and the floating-point execution units (adder, multiplier and divider) share a pipeline. The data cache is pseudo-dual ported via interleaving, with one port dedicated to loads and the other to stores. Most simple operations (such as integer ALU, floating-point add and floating-point multiply) can be pipelined with a throughput of one or two operations per clock cycle. The floating-point divider is not pipelined. Long latency operations can proceed in parallel with short latency operations.

The Pentium Pro processor pipeline contains three parts: (1) the in-order issue front-end, (2) the out-of-order core, and (3) the in-order retirement unit. Figure 2-3 details the entire Pentium Pro processor pipeline.

Optimizations!

- Processor Register and Functional Unit
- Out-Of-Order Execution
- Speculative Execution
- Branch Prediction
- 1 Pipeline, multiple execution units
 - i.e. Integer ALU and FPU (adder, multiplier and divider) share a pipeline
- Data cache pseudo-dual ported via interleaving
- “ Long latency operations can proceed in parallel with short latency operations.”

Optimizations!

- Processor Register and Functional Unit
- Out-Of-Order Execution
- Speculative Execution
- Branch Prediction
- 1 Pipeline, multiple execution units
 - i.e. Integer ALU and FPU (adder, multiplier and divider) share a pipeline
- Data cache pseudo-dual ported via interleaving
- “ Long latency operations can proceed in parallel with short latency operations.”

Optimizations!

L1 Data Cache Loads can:

- Read data before preceding stores when the load address and store address ranges are known not to Conflict.
- Be carried out speculatively, before preceding branches are resolved.
- Take cache misses out of order and in an overlapped manner.

Optimizati

TR/i

**YOU GET A SIDE CHANNEL,
AND YOU GET A SIDE CHANNEL**



EVERYONE GETS A SIDE CHANNEL

imgflip.com

Speculative Execution

- Bad speculation - The pipeline performs speculative execution of instructions that never successfully retire. The most common case is a branch misprediction where the pipeline predicts a branch target in order to keep the pipeline full instead of waiting for the branch to execute. If the processor prediction is incorrect it has to flush the pipeline without retiring the speculated instructions.

Speculative Execution

```
mov rax,[addr_0]
```

```
mov rbx,[addr_1]
```

Speculative Execution

```
mov rax,[addr_0]
add rax, 1
mov rbx,[rax + addr_1]
```

However, the second instruction will also execute speculatively and it may change the microarchitectural state of the CPU in a way that we can detect it. In this particular case the second mov instruction will load the someusermodeaddress into the cache hierarchy and we will be able to observe faster access time after structured exception handling took care of the exception.

Speculative Execution

```
mov rax,[addr_0]
```

```
add rax, 1
```

```
mov rbx,[rax + addr_1] ← Time Memory Load
```

However, the second instruction will also execute speculatively and it may change the microarchitectural state of the CPU in a way that we can detect it. In this particular case the second mov instruction will load the someusermodeaddress into the cache hierarchy and we will be able to observe faster access time after structured exception handling took care of the exception.

Speculative Execution

Uses: Arbitrary Kernel Memory Leak!

```
mov rax,[k_addr]
```

- Interrupt occurs
- Undefined behavior
 - Timing of finished instruction execution and actual retirement
 - mov potentially sets the results in the reorder buffer

Goal: Speculatively execute instructions after mov, based on reorder buffer value.

Speculative Execution

syscall	←	Force target into cache
mov rax,[k_addr]	←	Gussed address
add rax, 1		
mov rbx,[rax + addr_1]	←	Time to validate guess

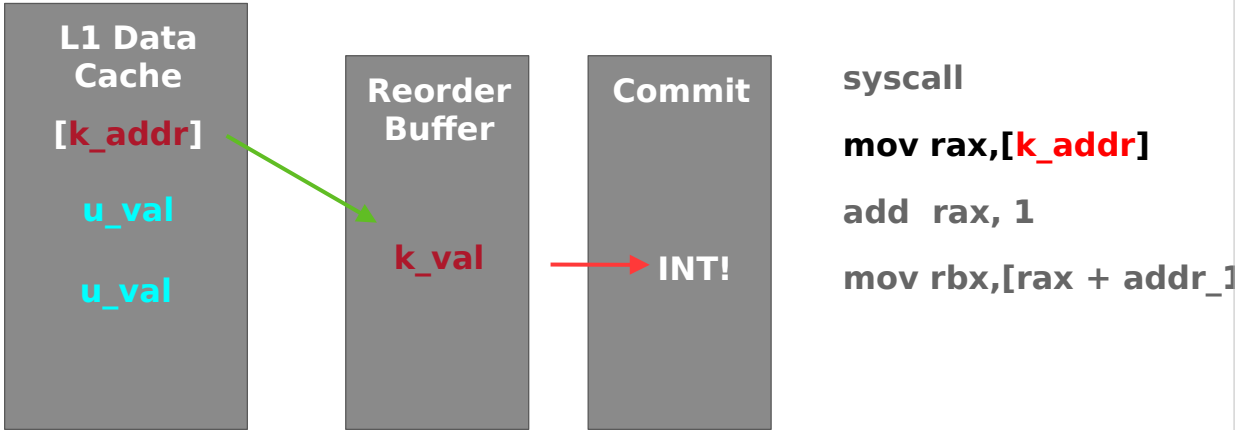
However, the second instruction will also execute speculatively and it may change the microarchitectural state of the CPU in a way that we can detect it. In this particular case the second mov instruction will load the someusermodeaddress into the cache hierarchy and we will be able to observe faster access time after structured exception handling took care of the exception.

Speculative Execution: Tomasulo algorithm



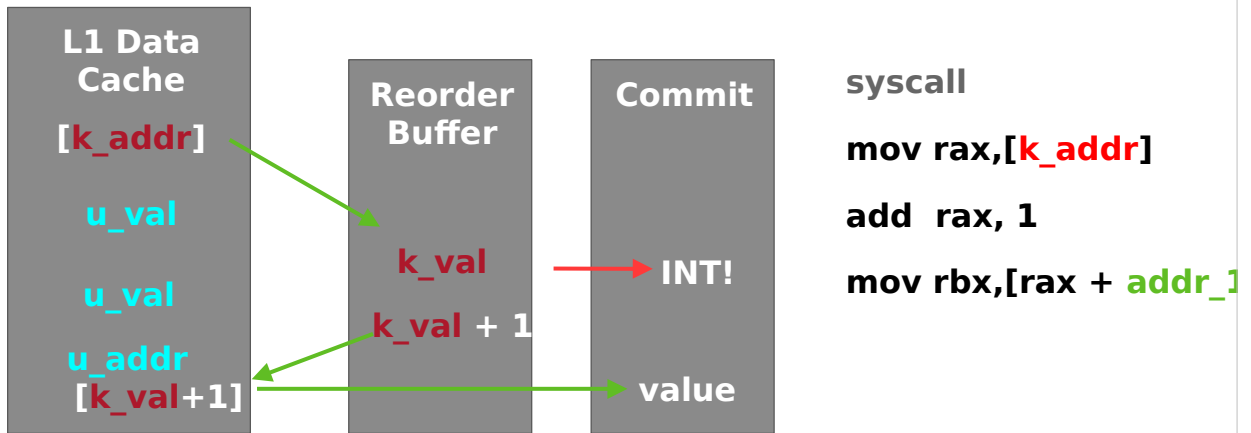
However, the second instruction will also execute speculatively and it may change the microarchitectural state of the CPU in a way that we can detect it. In this particular case the second `mov` instruction will load the `someusermodeaddress` into the cache hierarchy and we will be able to observe faster access time after structured exception handling took care of the exception.

Speculative Execution: Tomasulo algorithm



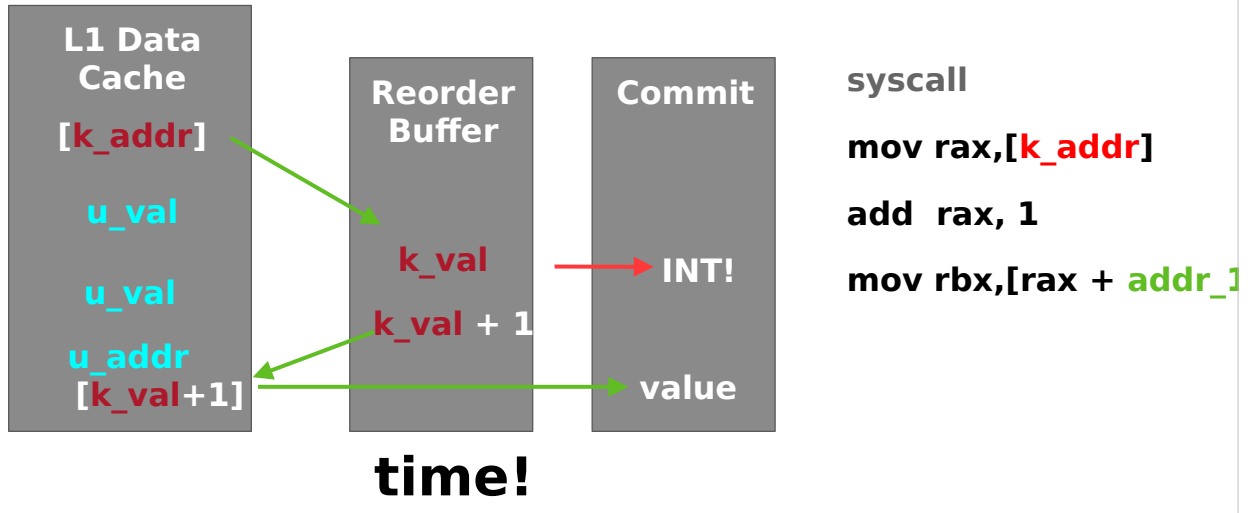
However, the second instruction will also execute speculatively and it may change the microarchitectural state of the CPU in a way that we can detect it. In this particular case the second `mov` instruction will load the `someusermodeaddress` into the cache hierarchy and we will be able to observe faster access time after structured exception handling took care of the exception.

Speculative Execution: Tomasulo algorithm



However, the second instruction will also execute speculatively and it may change the microarchitectural state of the CPU in a way that we can detect it. In this particular case the second `mov` instruction will load the `someusermodeaddress` into the cache hierarchy and we will be able to observe faster access time after structured exception handling took care of the exception.

Speculative Execution: Tomasulo algorithm



However, the second instruction will also execute speculatively and it may change the microarchitectural state of the CPU in a way that we can detect it. In this particular case the second `mov` instruction will load the `someusermodeaddress` into the cache hierarchy and we will be able to observe faster access time after structured exception handling took care of the exception.

Speculative Execution

Test target: Intel Broadwell CPU

- While goal k_addr value might not be given directly
- Use cache side channel to verify result or not
- Failed on this target, but..
 - Does process illegal read from k_addr (!)
 - Does not copy value into reorder buffer :<
 - Loads from data cache during speculative execution
 - Speculative execution & data loads do occur after violation of kernel/user read

Speculative Execution

Test target: Intel Broadwell CPU

- While goal k_addr value might not be given directly
- Use cache side channel to verify result or not
- Failed on this target, but..
 - Does process illegal read from k_addr (!)
 - Does not copy value into reorder buffer :<
 - Loads from data cache during speculative execution
 - Speculative execution & data loads do occur after violation of kernel/user read

To be continued....

Acknowledgements

co-author: Jeremy Blackthorne
advisor: Bulent Yener

Trail of Bits
Ryan Stortz, Jeff Preshing, Anders Fogh

<https://www.sophia.re/SC>

<http://preshing.com/20120515/memory-reordering-caught-in-the-act/>

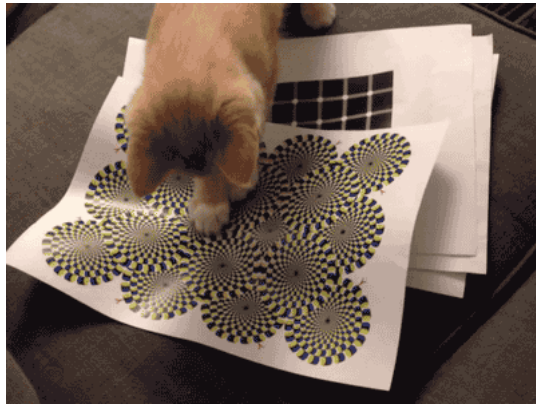
<http://blog.stuffedcow.net/2014/01/x86-memory-disambiguation/>

<https://cyber.wtf/2017/07/28/negative-result-reading-kernel-memory-from-user-mode/>

<https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf>



Any Questions?



IRC: quend

email: sophia@trailofbits.com

website: www.sophia.re