Computer Architecture. Week 11

Muhammad Fahim, Alexander Tormasov

Innopolis University

m.fahim@innopolis.ru
a.tormasov@innopolis.ru

November 12, 2020

Topic of the lecture

- Overview of Modern Computers (Vulnerabilities)

Topic of the tutorial

- Control signals of Single Cycle Processor

Topic of the lab

- Development of Single Cycle Processor

Content of the class

- Vulnerabilities and Exploits
- Security Goals
- Meltdown and Spectre
- Threat-o-meter
- Memory model and process
- Side channeling
- Execution order – out-of-order execution
- Speculative execution
- Summary

Vulnerabilities and Exploits

- Vulnerability is a weakness in a system.
- An exploit is an attack that leverages that vulnerability.
- So vulnerable means there is theoretically a way to exploit something (i.e., a vulnerability exists), exploitable means that there is a definite path to doing so in the wild.
- Naturally, attackers want to find weaknesses that are actually exploitable.

Vulnerable CPU

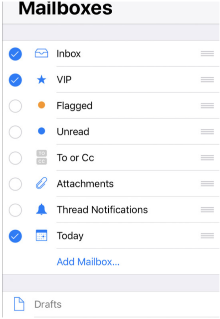
- The current set of vulnerabilities exploit how our modern processors work.
- Unlike software-based attacks, these hardware vulnerabilities allow programs to steal data which is currently processed on the computer.
- So, the problem is very low level and uses the actual CPU hardware architecture.

Security Goals

- A process must be isolated (Protected) from other processes!

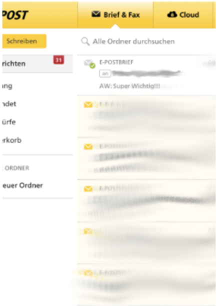
Security Goals: App Isolation

You don't want *this*



To read *that*

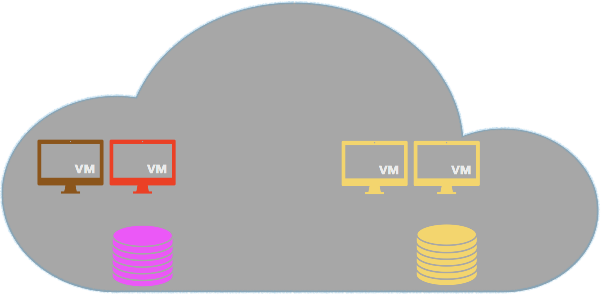
Security Goals: Browser Tab Isolation



You don't want *this*

To read *that*

Security Goals: Cloud Isolation



You don’t want *this* To read *that*

Security Goals: Software vulnerabilities

- The most common software security vulnerabilities include:
  - Missing data encryption
  - OS command injection
  - SQL injection
  - Buffer overflow
  - Missing authentication for critical function
  - Missing authorization
  - Unrestricted upload of dangerous file types
  - Reliance on untrusted inputs in a security decision
  - Cross-site scripting and forgery
  - Download of codes without integrity checks
  - Use of broken algorithms
  - URL redirection to untrusted sites
  - Path traversal
  - Bugs
  - Weak passwords
  - Software that is already infected with virus

Security Goals: Memory Isolation (1/2)

PROCESS A

PROCESS B

You don't want *this* | To read *that*

You don’t want *this*

To read *that*

Security Goals: Memory Isolation (2/2)

- The more a system relies on process isolation to achieve its security goal, the more critical Meltdown and Spectre are.
- Fortunately an attacker must be able to **execute his code on a system** to exploit the Meltdown and Spectre attacks.

Security Goals: Cold Boot Attacks

On February 2008...

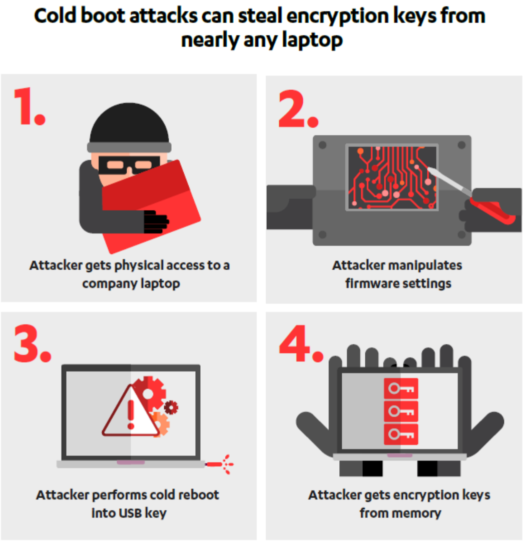
Hackers Decrypt Computer by Freezing Memory Chips

TECHNOLOGY

Researchers Find Way to Steal Encrypted Data

Security Goals: Cold Boot Attacks

- Memory modules gradually lose data over time as they lose power, but do not immediately lose all data when power is lost
- Depending on temperature and environmental conditions, memory modules can potentially retain, at least, some data for up to 90 minutes after power loss.
- With certain memory modules, the time window for an attack can be extended to hours or even weeks by cooling them with freeze spray.
- Liquid nitrogen, freeze spray or compressed air cans can be improvised to cool memory modules, and thereby slow down the degradation of volatile memory


Cold boot attacks can steal encryption keys from nearly any laptop
1.
Attacker gets physical access to a company laptop
2.
Attacker manipulates firmware settings
3.
Attacker performs cold reboot into USB key
4.
Attacker gets encryption keys from memory

Security Goals: Cold Boot Attacks

- A cold boot attack is a type of side channel attack in which an attacker with physical access to a computer performs a memory dump of a computer's random access memory by performing a hard reset of the target machine.
- Typically, cold boot attacks are used to retrieve encryption keys from a running operating system for malicious or criminal investigative reasons.
- The attack relies on the data remanence property of DRAM and SRAM to retrieve memory contents that remain readable in the seconds to minutes after power has been removed

Foreshadow (Security vulnerability)

- Foreshadow is a vulnerability that affects modern microprocessors that was first discovered in January 2018.
- The vulnerability is a speculative execution attack on Intel processors that may result in the disclosure of sensitive information stored in personal computers and third-party clouds.

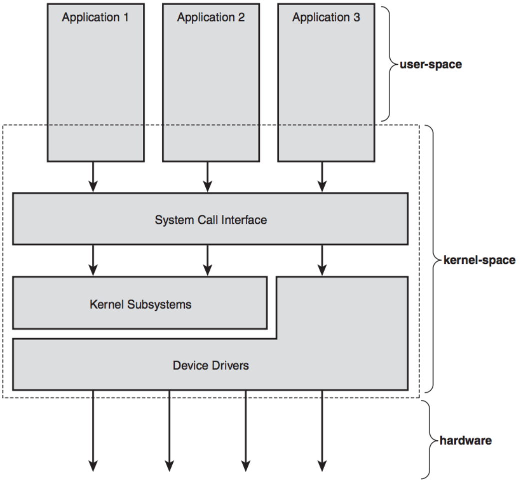
Meltdown and Spectre

- Two security flaws were unveiled by security researchers (3rd January 2018)
  - Meltdown
  - Spectre
- At Google's Project Zero in conjunction with academic and industry researchers from several countries

Meltdown and Spectre

- The vulnerabilities behind the devastating Meltdown and Spectre attacks have existed for decades.
- Meltdown is a massive vulnerability on nearly every Intel chip made since 1995, but is largely being fixed with software patches.
- Spectre is more difficult to exploit, but will likely be with us for years
- Both exploits allow a malicious user to access data, whether that's your password, credit-card number, or just your personal photos stored on a cloud server.

Memory – User and Kernel Spaces



- NOTE: More details in Operating System (OS) course.

The Attacker Code – Example

```
; rcx = kernel address
; rbx = probe array
retry:
mov al, byte [rcx]
shl rax, 0xc
jz retry
mov rbx, qword [rbx + rax]
```

- **Explanation:** An inaccessible kernel address is moved to a register, raising an exception. The subsequent instructions are already executed out of order before the exception is raised, leaking the content of the kernel address through the indirect memory access.

Meltdown

- **Result:** Programs can read memory it should not
- **Affects:** All modern CPU/OS
- **Vector:** Uses out of order execution to read forbidden memory and cache timing as side channel to exfiltrate data
- **How bad:** Bad
- **Fixes:** Needs changes in CPU and/or OS patches. Modest (X%) to severe (XX%) performance impact, higher on older CPU. Performance impact varies and depends on CPU and workload type.

Spectre

- **Result:** Programs can read all memory
- **Affects:** All modern CPU/OS
- **Vector:** Uses speculative execution to read forbidden memory and cache timing to exfiltrate data
- **How bad:** Very bad
- **Fixes:** Needs changes in CPU and/or changes in programs. Performance impact varies and depends on CPU and workload type.

Threat-o-meter

Exploit unlikely or running untrusted code already worst case

Exploit possible but needs another successful attack to run attackers code

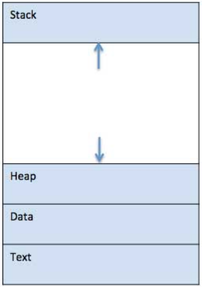
Exploit possible and runs untrusted code "by design"

Process

- A process is an instance of a computer program that is being executed.
- It contains the program code and its activity.
- Depending on the Operating System (OS), a process may be made up of multiple threads of execution that execute instructions concurrently.

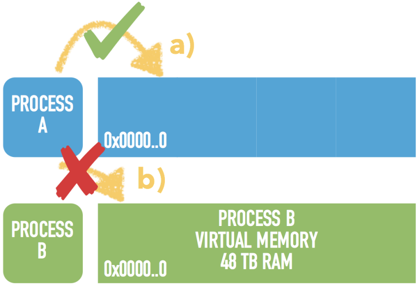
Process – Components

- When a program is loaded into the memory and it becomes a process, it can be divided into four sections – stack, heap, text and data.



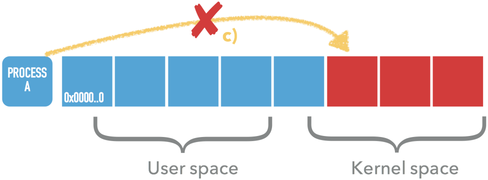
NOTE: More details in OS course.

Memory Model (1/5)



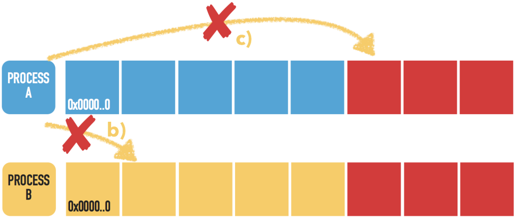
- CPU and OS isolate processes memory from each other
- Virtual Memory gives each process its own address space
- Each address space starts at “virtual address 0x000..0”

Memory Model (2/5)



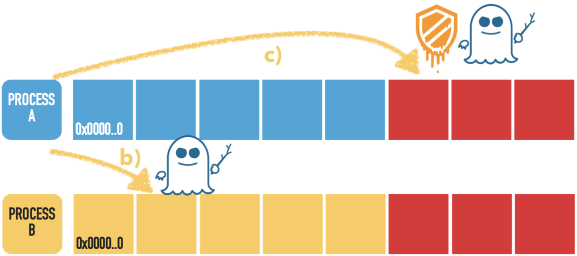
- Memory is split into pages (each 4KiB on x86)
- The kernel maps its own memory into each process
- This "kernel" memory is only accessible by the kernel

Memory Model (3/5)



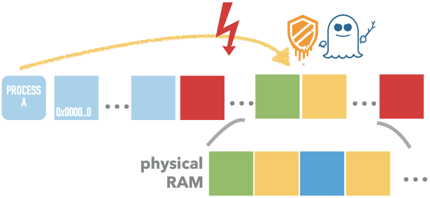
- b) and c) are completely different error scenarios
  - c) Kernel memory pages are marked "kernel only" but the process could try to access the pages via a pointer
  - b) Process b has no possibility to even describe the address

Memory Model (4/5)



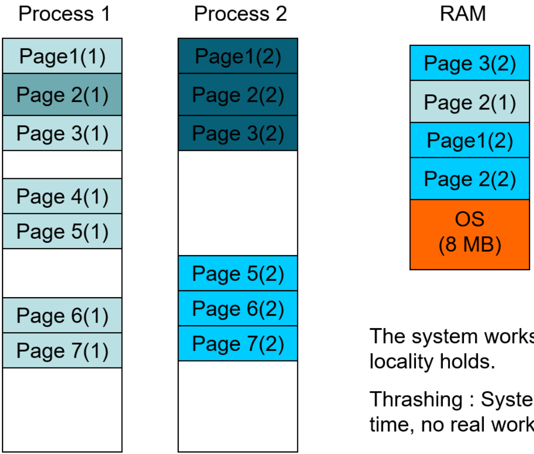
- c) is vulnerable to Meltdown and Spectre
- b) is vulnerable to Spectre

- Virtual memory is backed by physical RAM
- Virtual memory is much, much larger than physical RAM
- Not all virtual memory is backed by RAM



- Like a matryoshka doll the kernel maps all physical memory into its address space
- Reading kernel memory allows reading of all (mapped) memory of all processes

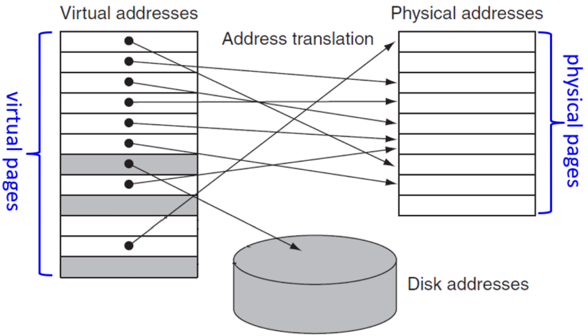
Memory Model – Virtual Memory (2/3)



The system works because principle of locality holds.

Thrashing : System swaps in/out all the time, no real work is done.

Memory Model – Virtual Memory (2/3)



In virtual memory, blocks of memory (called pages) are mapped from one set of addresses (called virtual addresses) to another set (called physical addresses)

Memory Model (5/5)

Virtual memory map with 4 level page tables:

```
   Start addr    |   Offset   |     End addr     |  Size   | VM area description
========================================================================================================================
                 |            |                  |         |
 0000000000000000 |    0       | 00007fffffffffff |  128 TB | user-space virtual memory, different per mm
__________________|____________|__________________|_________|___________________________________________________________
                 |            |                  |         |
 0000800000000000 | +128    TB | ffff7fffffffffff | ~16M TB | ... huge, almost 64 bits wide hole of non-canonical
                 |            |                  |         |     virtual memory addresses up to the -128 TB
                 |            |                  |         |     starting offset of kernel mappings.
__________________|____________|__________________|_________|___________________________________________________________
                                                           |
                                                           | Kernel-space virtual memory, shared between all processes:
____________________________________________________________|___________________________________________________________
                 |            |                  |         |
 ffff800000000000 | -128    TB | ffff87ffffffffff |    8 TB | ... guard hole, also reserved for hypervisor
 ffff880000000000 | -120    TB | ffffc7ffffffffff |   64 TB | direct mapping of all physical memory (page_offset_base)
 ffffc80000000000 |  -56    TB | ffffc8ffffffffff |    1 TB | ... unused hole
 ffffc90000000000 |  -55    TB | ffffe8ffffffffff |   32 TB | vmalloc/ioremap space (vmalloc_base)
 ffffe90000000000 |  -23    TB | ffffe9ffffffffff |    1 TB | ... unused hole
 ffffea0000000000 |  -22    TB | ffffeaffffffffff |    1 TB | virtual memory map (vmemmap_base)
 ffffeb0000000000 |  -21    TB | ffffebffffffffff |    1 TB | ... unused hole
 ffffec0000000000 |  -20    TB | fffffbffffffffff |   16 TB | KASAN shadow memory
 fffffc0000000000 |   -4    TB | fffffdffffffffff |    2 TB | ... unused hole
                 |            |                  |         | vaddr_end for KASLR
 fffffe0000000000 |   -2    TB | fffffe7fffffffff |  0.5 TB | cpu_entry_area mapping
 fffffe8000000000 |   -1.5  TB | fffffeffffffffff |  0.5 TB | LDT remap for PTI
 ffffff0000000000 |   -1    TB | ffffff7fffffffff |  0.5 TB | %esp fixup stacks
__________________|____________|__________________|_________|____________________________________________________________
                                                           |
                                                           | Identical layout to the 47-bit one from here on:
____________________________________________________________|____________________________________________________________
                 |            |                  |         |
 ffffff8000000000 | -512    GB | ffffffeeffffffff |  444 GB | ... unused hole
 ffffffef00000000 |  -68    GB | fffffffeffffffff |   64 GB | EFI region mapping space
 ffffffff00000000 |   -4    GB | ffffffff7fffffff |    2 GB | ... unused hole
 ffffffff80000000 |   -2    GB | ffffffff9fffffff |  512 MB | kernel text mapping, mapped to physical address 0
 ffffffff80000000 |-2048    MB |                  |         |
 ffffffffa0000000 |-1536    MB | fffffffffeffffff | 1520 MB | module mapping space
 ffffffffff000000 |  -16    MB |                  |         |
    FIXADDR_START | ~-11    MB | ffffffffff5fffff | ~0.5 MB | kernel-internal fixmap range, variable size and offset
 ffffffffff600000 |  -10    MB | ffffffffff600fff |    4 kB | legacy vsyscall ABI
 ffffffffffe00000 |   -2    MB | ffffffffffffffff |    2 MB | ... unused hole
__________________|____________|__________________|_________|___________________________________________________________
```

https://www.kernel.org/doc/Documentation/x86/x86_64/mm.txt

Execution Order

- Animated Slides (Power point)

- Animated Slides (Power point)

Summary

- Vulnerabilities and Exploits
- Meltdown and Spectre
- Memory model and process Virtual Memory
- Side channeling
- Execution order – out-of-order execution
- Speculative execution

Acknowledgment

- IA-32 Intel: Architecture Software Developer's Manual Volume 3: System Programming Guide (Document 253668): Chapter 3 4.
- Meltdown Spectre for normal people (`https://github.com/neuhalje`)
- White paper on Meltdown Spectre (`http://cert-mu.govmu.org/English/Documents/White%20Papers/MELTDOWN%20-%20CERTMU%20WHITEPAPER.pdf`)