Meltdown: Reading Kernel Memory from User Space

Independently discovered and reported by three teams:

Jann Horn (Google Project Zero)
Werner Haas, Thomas Prescher (Cyberus Technology),
Daniel Gruss, Moritz Lipp, Stefan Mangard, Michael Schwarz (Graz University of Technology)
You may have already known...

Major chip flaws affect billions of devices
by Selena Larson @CNNTech

January 4, 2018, 9:44 AM ET

Two major flaws in computer chips could leave a huge number of computers and smartphones vulnerable to security concerns, researchers revealed Wednesday.

And a U.S. government-backed body warned that the chips themselves need to be replaced to completely fix the problems.
Executive Summary of Meltdown

- **Observation:** Out-of-order execution allows access of invalid memory address before checking the validation.

- **Attack description (briefly):**
  - Raise exception before accessing an invalid address.
  - Out-of-order execution causes microarchitectural change.
  - Use side-channel attack to recover the secret.

- **Mitigation:** KAISER -- kernel address isolation to have side-channels efficiently removed
Outline

- Introduction
- Background
  - Out-of-order execution
  - Address spaces
- Meltdown attack
- Countermeasure
- Evaluation
- Strengths and weaknesses
- Discussion
What is Meltdown?

Meltdown breaks the most fundamental isolation between user applications and the operating system. This attack allows a program to access the memory, and thus also the secrets, of other programs and the operating system.
Introduction

- How is it different from other attacks?
  No software vulnerability
  Exploit side-channel information

- Which systems are affected by Meltdown?
  Every Intel processor which implements out-of-order execution since 1995.
Outline

- Introduction
- Background
  - Out-of-order execution
  - Address spaces
- Meltdown attack
- Countermeasure
- Evaluation
- Strengths and weaknesses
- Discussion
Background – Out-of-order execution

- Out-of-order execution:
  - Optimization technique
  - CPU executes instructions as soon as all required resources are available.

- In practice, running operations *speculatively* before the CPU is certain whether the instruction will be needed and committed.
Background – Out-of-order execution

- Intel Architecture
  - Reorder buffer: register allocation, register renaming, and retiring.
  - Unified reservation station: queues the operations on exit ports that are connected to Execution Units
- Tomasulo Algorithm
Virtual address space: virtual addresses are translated to physical addresses to isolate processes from each other

Virtual space is split into a user and a kernel space

The entire physical memory is typically mapped in the kernel space

- Direct map: Linux and OS X
- Paged pool, non-paged pool and system cache: Windows
In order to protect the kernel from memory corruption bugs, address space layout randomization (ASLR) has been introduced.

KASLR randomizes the offsets where drivers are located on every boot.
- Still not sufficient to prevent all attacks

Solution to KASLR attacks (KAISER) solves the Meltdown Attack as well!
Outline

- Introduction
- Background
  - Out-of-order execution
  - Address spaces
- Meltdown attack
- Countermeasure
- Evaluation
- Strengths and weaknesses
- Discussion
Meltdown – A toy example

- Let’s first look at a code snippet
- In theory: cannot access the array
- In reality: may have already executed instructions

```c
1    raise_exception();
2    // the line below is never reached
3    access(probe_array[data * 4096]);
```
Meltdown – A toy example

```
probe_array[data*4096]
probe_array[data*4096]
```
Meltdown – A toy example

Register is cleared

But cache state remains!
Meltdown – A toy example
Meltdown – Building Block

transient instruction: executed out of order and leaving measurable side effects
Issue in Executing Transient Instructions

- Reason: prevent the process from being killed

- Exception handling:
  - Fork the attacking application before accessing the invalid memory location
  - Install a signal handler that is executed when a certain exception occurs

  🔄 Reducing performance overhead
Exception suppression:
  - Transactional memory

Put the invalid memory access after a never-taken branch:
  - Setup phase: Mistrain CPU into speculatively executing these instructions.
  - Second phase: speculatively execute an instruction that leak information
  - Final phase: Recover data by retrieving over covert. channel
Issue in Building a Covert Channel

- Sending end: the transient instruction sequence
- Receiving end: can be a different thread or even a different process

- The covert channel is not limited to rely on cache:
  - ALU contention

- But here we use Flush+Reload cache attack
Meltdown consists of 3 steps:

- The content of an attacker-chosen memory location, which is inaccessible to the attacker, is loaded into a register.

- A transient instruction accesses a cache line based on the secret content of the register.

- The attacker uses Flush+Reload to determine the accessed cache line and hence the secret stored at the chosen memory location.
Outline

- Introduction
- Background
  - Out-of-order execution
  - Address spaces
- Meltdown attack
- Countermeasure
- Evaluation
- Strengths and weaknesses
- Discussion
Countermeasures – Hardware

- Trivial solution: completely disable out-of-order execution
  - The performance impacts would be devastating

- Serializing the permission check and the register fetch can prevent Meltdown attack.
  - This involves a significant overhead to every memory fetch

- Introduce a hard split of user space and kernel space
  - Expect minimal performance impacts

- Note: the above methods only solve Meltdown, not Spectre
Countermeasures – Software (KAISER)

- A kernel modification that does not have the kernel mapped in user space.
- Reason: no valid mapping to kernel space or physical memory available in user space.
- However, there exists a residual attack surface for Meltdown.
- Still, the best short-time solution currently available.
Outline

- Introduction
- Background
  - Out-of-order execution
  - Address spaces
- Meltdown attack
- Countermeasure
- Evaluation
- Strengths and weaknesses
- Discussion
Evaluation– Sample Code

- Assembly code given by the paper:

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

Look not so nice…
Evaluation – Sample Code

- A more readable one:

```c
char detectArray[256*64];
flush_array(detectArray);

try{
    char a = *kernel_address;
    detectArray[a*64] ++;
}catch(segfault){}

for(int i=0; i<256; i++){
    if(is_in_cache(detectArray[i*64]))
        printf("secret byte was %d\n", i);
}
```
Evaluation – Environment

- Tested platform:
  - Linux without KAISER ✓
  - Windows 10 without KAISER ✓
  - Linux with KAISER ×
  - Containers such as Docker ✓
  - Android (ARM) ×

<table>
<thead>
<tr>
<th>Environment</th>
<th>CPU Model</th>
<th>Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lab</td>
<td>Celeron G540</td>
<td>2</td>
</tr>
<tr>
<td>Lab</td>
<td>Core i5-3230M</td>
<td>2</td>
</tr>
<tr>
<td>Lab</td>
<td>Core i5-3320M</td>
<td>2</td>
</tr>
<tr>
<td>Lab</td>
<td>Core i7-4790</td>
<td>4</td>
</tr>
<tr>
<td>Lab</td>
<td>Core i5-6200U</td>
<td>2</td>
</tr>
<tr>
<td>Lab</td>
<td>Core i7-6600U</td>
<td>2</td>
</tr>
<tr>
<td>Lab</td>
<td>Core i7-6700K</td>
<td>4</td>
</tr>
<tr>
<td>Lab</td>
<td>Core i7-8700K</td>
<td>12</td>
</tr>
<tr>
<td>Lab</td>
<td>Xeon E5-1630 v3</td>
<td>8</td>
</tr>
<tr>
<td>Cloud</td>
<td>Xeon E5-2676 v3</td>
<td>12</td>
</tr>
<tr>
<td>Cloud</td>
<td>Xeon E5-2650 v4</td>
<td>12</td>
</tr>
<tr>
<td>Phone</td>
<td>Exynos 8890</td>
<td>8</td>
</tr>
</tbody>
</table>
Evaluation – Performance

- With exception handling:
  - More universal implementation
  - Achieve an average reading speed of 123 KB/s when leaking 12 MB of kernel memory
  - Error rate of 0.03 %
  - Channel capacity is 122 KB/s
Evaluation – Performance

- With exception suppression:
  - Conditional branches or Intel TSX
  - Achieve an average reading speed of 503 KB/s when leaking 12 MB of kernel memory
  - Error rate of 0.02 %
  - Channel capacity is 502KB/s
Evaluation – In Practice

- Memory dump showing HTTP Headers on Ubuntu 16.10 on a Intel Core i7-6700K

- The XX cases represent bytes where the side channel did not yield any results
Memory dump of Firefox 56 on Ubuntu 16.10 on a Intel Core i7-6700K disclosing saved passwords
Evaluation – Limitation

- They did not manage to successfully leak kernel memory with the meltdown attack neither on ARM nor on AMD.

- Reasons:
  - The implementation might simply be too slow
  - Processor lacks certain features

- However, the toy example works reliably.
Executive Summary of Meltdown

- **Observation**: Out-of-order execution allows access of invalid memory address before checking the validation.

- **Attack description (briefly)**:
  - Raise exception before accessing an invalid address.
  - Out-of-order execution causes microarchitectural change.
  - Use side-channel attack to recover the secret.

- **Mitigation**: KAISER -- kernel address isolation to have side-channels efficiently removed
Outline

- Introduction
- Background
  - Out-of-order execution
  - Address spaces
- Meltdown attack
- Countermeasure
- Evaluation
- Strengths and weaknesses
- Discussion
The paper presented a potential attack on a wide range of modern processors which could cause catastrophic problems.

This attack didn’t exploit any software vulnerability and therefore can be launched in all operating systems.

The paper gave both short-term software solution and long-term hardware solution and verified the former’s effectiveness.
Weaknesses

- They included too much background information and made the paper not easy to read.

- They didn’t give a practical attack on platforms other than Intel and didn’t know the exact reason.

- They didn’t propose a better software solution.

- They didn’t evaluate the performance impacts by the KAISER patch.
Takeaway

- From a computer security perspective:
  - Attacks can happen at any level – previously we’ve seen memory performance attack and Row-hammer attack
  - Knowledge in computer architecture can aid security professions to find out new “bugs”
  - Covert channel (side channel) is a fascinating topic

Onur Mutlu, Computer Architecture Lecture 1, Fall 2018
Takeaway

- From a computer architecture perspective:
  - Design new architecture with a high-security guarantee at the very beginning
  - Balance cost, performance, and security when designing
  - Look back at the architecture from time to time in order to look for new faults
Meltdown: Reading Kernel Memory from User Space

Questions?
Open discussion question

Timing Attacks on RSA: Revealing Your Secrets through the Fourth Dimension

by Wiro H. van Herk

Introduction

If Alice wants to secure her home, she could buy high-quality locks and install several of them on her door. However, a clever burglar might simply unscrew the hinges, remove the door and walk away with all of Alice’s valuables with minimal effort. This example of an indirect attack on household security is somewhat artificial but there exists a parallel in the world of encryption that is quite real. It is called the timing attack and it has been used to defeat some of the most popular encryption techniques.

Do you think your computer system is secure because you use strong cryptography? Do you know attackers may be able to attack your cryptography in a completely unexpected direction without directly breaking the cryptographic algorithm?

RSA (7) is a public key cryptographic algorithm that is widely used today to secure electronic data transfer. It is included as part of the Web browser from Microsoft and Netscape and is used by the SSL (Secure Sockets Layer) which provides security and privacy over the Internet. The RSA algorithm was invented by the team of Rivest, Shamir, and Adleman at MIT in 1978. Independently, Diffie and Hellman discovered the same idea in the early 1970's [5]. A public key cryptosystem uses a one-way function that is easy to compute in one direction and hard to compute in the reverse direction. For example, it is relatively easy to generate two prime numbers $p$ and $q$ and compute their product $N = p q$. But given $N$, it is difficult to find its factors $p$ and $q$. Encryption uses a public value, or key, which is distributed and known to anyone who wants to send a message. Decryption involves a related private key which is kept secret by the intended recipient and cannot be deduced from the public key. Public key cryptography works without requiring both parties involved to keep an agreed upon secret; the private key never needs to be sent to the sender.

RSA’s public key includes a number $N$ which is the product of two large prime numbers $p$, $q$. The strength of RSA comes from the fact that factoring large numbers is difficult. The best-known factoring methods are still very slow. For example, in a recent RSA challenge (August 1999), a 512-bit RSA challenge number was factored using 292 workstations and high-speed computers. The factoring took 19.7 CPU-years to accomplish which is equivalent to approximately 80,000 MIPS years. The feat required 3.7 months of calendar time [5]. Because so many people have been trying to find efficient ways to factor large numbers, so far with great success, we can probably assume that RSA is safe from a factoring attack for a typical key size of 1024 bits in length. RSA can be made more secure against factoring by increasing the key length to 2048 bits or more.

Despite this formidable mathematical strength, research has shown that it is feasible to recover RSA private keys without directly breaking RSA. This type of attack is known as a timing attack in which an attacker observes the running time of a cryptographic algorithm and thereby deduces the secret parameter involved in the operations. It is generally agreed that RSA is secure from a direct attack. RSA’s vulnerability to timing attacks is not so well known and often overlooked. This paper explains the principles of timing attacks on RSA, summarizes the results of timing attacks implemented by various researchers, and discusses defenses against such attacks.

Timing Attacks on RSA

Kocher [4] was the first to discuss timing attacks. At the RSA Data Security and CRYPTO conferences in 1996, Kocher presented his preliminary result, warned vendors about his attack, and caught the attention of cryptographers including the inventors of the RSA cryptosystem. Timing attacks are a form of side channel attack where an attacker gains information from the implementation of a cryptosystem rather than from any inherent weakness in the mathematical properties of the system. Unintended channels of information arise due to the way an operation is performed or the media used. Side channel attacks exploit information about timing, power consumption, electromagnetic emanations or even sound to recover secret information about a cryptosystem [6].

Timing attacks exploit the timing variations in cryptographic operations. Because of performance optimizations, computations performed by a cryptographic algorithm often take different amounts of time depending on the input and the value of the secret parameter. If RSA private key operations can be timed reasonably accurately, in some cases statistical analysis can be applied to recover the secret key involved in the computations.

Before discussing timing attacks on RSA, we must first consider the mathematics of the cryptosystem. RSA is a public key cryptosystem that uses a public exponent $e$ for encryption and a private exponent $d$ for decryption. It uses a modulus $N$ which is a product of two large prime numbers $p$ and $q$; i.e., $N = p q$. The exponents $e$ and $d$ must be chosen to satisfy the condition $ed - 1 = 1 \mod (p - 1)(q - 1)$. Then the RSA key pair consists of the public key $(N, e)$ and the private key $(N, d)$. For example, if we select two prime numbers $p = 11$ and $q = 3$, then $N = 11 \times 3 = 33$. Now compute $(p - 1)(q - 1) = 10 \times 2 = 20$ and choose a value of $e$ relatively prime to 20, say 3. Then choose to be chosen such that $ed = 1 \mod 20$. For example, $e = 3$ since $3 \times 3 = 9 = 1 \mod 20$. So we pick the public key $(33, 3)$ and the corresponding private key $d = 7$. We disclose the rational factors of $n$...
Open discussion question

TLBLEED

Overview

TLBleed is a new side channel attack that has been proven to work on Intel CPU’s with Hyper-threading (generally Simultaneous Multi-threading, or SMT, or HT on Intel) enabled. It relies on concurrent access to the TLB, and it being shared between threads. We find that the L1dtlb and the STLB (L2 TLB) is shared between threads on Intel CPU cores.

Result

This means that co-resident hyperthreads can get a certain amount of information on the data memory accesses of the other HT, without needing any shared cache. Whereas the cache can be partitioned to protect against cache attacks (such as in Cloak using TSX, or using CAT, or using page coloring), this is practically impossible for the TLB, and no such systems have been proposed. Thus in the presence of cache defenses, TLBleed remains a risk in this threat model.

Requirements (a.k.a. threat model)

The threat model is identical to Colin Pervical’s seminal 2005 work, Cache Missing for Fun and Profit, which arguably introduced practical cache side channels. Different from TLBleed, it requires concurrent access to the cache shared between Hyperthreads. TLBleed gets all information through the shared TLB, which can’t be partitioned between processes in software (either application or OS).
Meltdown Attack Lab

- **Website**

- Hands-on experience and very detailed instruction
Useful Resources

- YouTube video of attack demos:
  - https://youtu.be/L1N1P2zxaZE
  - https://youtu.be/kwnh7q356Jk

- Recommended papers:
  - FLUSH+RELOAD: a High Resolution, Low Noise, L3 Cache Side-Channel Attack
  - KASLR is Dead: Long Live KASLR
  - Breaking Kernel Address Space Layout Randomization with Intel TSX
Thanks for your listening!

Meltdown: Reading Kernel Memory from User Space
Meltdown: Reading Kernel Memory from User Space

Supplementary Slides
Meltdown – Optimization

- **Case of 0:**
  - If the exception is triggered while trying to read from an inaccessible kernel address, the register where the data should be stored, appears to be zeroed out.
  - If the zeroing out of the register is faster than the execution of the subsequent instruction, the attacker may read a false value in the third step.
  - Meltdown retries reading the address until it encounters a value different from ‘0’.
  - Meltdown assumes that the secret value is indeed ‘0’ if there is no cache hit at all.

```plaintext
1 ; rcx = kernel address, rbx = probe array
2 xor rax, rax
3 retry:
4 mov al, byte [rcx]
5 shl rax, 0xc
6 jz retry
7 mov rbx, qword [rbx + rax]
```
Single-bit transmission:
- The performance bottleneck in the generic attack is Flush+Reload
- By transmitting only one bit, we only have to perform one Flush+Reload at one time.

Drawback: our side channel has a bias towards a secret value of ‘0’.
- The number of bits read and transmitted at once is a tradeoff between some implicit error-reduction and the overall transmission rate of the covert channel.
Meltdown – Optimization

- Dealing with KASLR:
  - With KASLR, the direct-physical map is randomized and not fixed at a certain address.
  - Need to obtain the randomized offset before mounting the Meltdown attack.
  - However, the randomization is limited to 40 bit – we can find out the randomized address quickly.
Background – Out-of-order execution

- Tomasulo Algorithm
- Enable dynamic scheduling and out-of-order execution
- Introduce a unified reservation station that allows a CPU to use a data value as it has been computed instead of storing it to a register and re-reading it
- Solve RAW, WAR, WAW hazards

All execution units are connected via a common data bus. The reservation units listen on the data bus.
Exploit timing differences that are introduced by the caches.

- Evict+Time, Prim+Probe, and Flush+Reload

We use Flush+Reload: exploits the shared, inclusive last-level cache

- Frequently flush a targeted memory location (clflush)
- Measure the time it takes to reload the data
- Determine whether data was loaded into the cache by another process

Building a covert channel to leak information from one security domain to another.