Spectre & Meltdown

Warum diese Sicherheitslücken eine solche Medienwirksamkeit erfahren haben

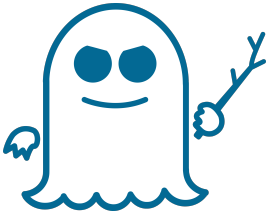
Thomas Kittel
Lehrstuhl für Sicherheit in der Informatik

21. März 2018

- Thomas Kittel
- Mitarbeiter am Lehrstuhl für Sicherheit in der Informatik der TU München.
- u.A. mitverantwortlich für Vorlesungen IT-Sicherheit und Sichere Mobile Systeme.

- Interessensschwerpunkte:
  - Betriebssystemsicherheit und Netzwerksicherheit.
  - Vorträge zum Themenbereich Datenschutz und Schutz vor Überwachung im Internet

Spectre



Meltdown

- Anfang des Jahres wurden zwei Sicherheitsprobleme durch die Medien getrieben.
  - Alle Prozessoren von Intel sollen defekt sein.
  - Vergleiche mit dem Dieselskandal in aller Munde.
  - Aber warum? Dieser Vortrag soll über die Gründe und die Schwachstellen aufklären.

- **Umfrage:** Wieviel technisches Vorwissen ist vorhanden?

Klassische Angriffe

- Phishing: Versuch mit gefälschter Email / Webseite Zugangsdaten abzufangen.
- Social Engineering: Sicheres Auftreten - Problemfaktor Mensch
- Schadsoftware (Viren, Malware) - Beispiele:
  - Bösartiges PDF, ZIP oder Word Dokument als Anhang einer Email.
  - Bösartiger USB Stick, der Computer beim Einstecken kapert.

Klassische Angriffe
(Remote) Code Execution

- Inkorrekte Programmausführung durch ungültige / unerwartete Eingaben.
- Fehlende Validierung der Eingaben
- Idealerweise von “außen“ über das Netzwerk

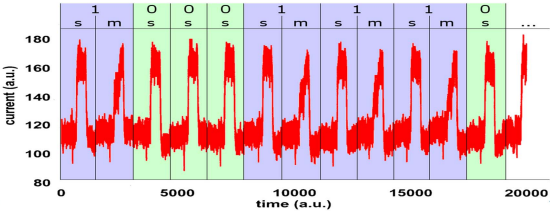
- Eingabe ist zu lange und überschreibt Daten im Arbeitsspeicher
- Programmierer erwartet einen Zustand, der Angreifer schafft es den Zustand zu verändern.
- Erweiterung von Rechten (User wird Administrator) oder
- Ausführen von unberechtigtem Programmcode.
- Unauthenticated Remote Code Execution als schlimmste Schwachstelle.

Seitenkanal Angriffe

- Ziel: Angreifer versucht Daten zu extrahieren.
- Aber: Der Angreifer kann die Daten nicht direkt lesen / senden.
- Idee: Verwende unabhängige Ereignisse zur Kommunikation (z.B. Lampe an - Lampe aus)

- Extraktion von Informationen über einen geheimen schmalbandigen Informationskanal.
- Ursprünglich im Bereich der eingebetteten Systeme.

- Washington Domino Pizza Index
  - Pizzaservice erkennt anhand der Bestellungen, dass die Regierung etwas plant.
- Stromverbrauch des Prozessors lässt Rückschlüsse auf Daten zu.
  - RSA Verschlüsselung: Square & Multiply am Stromverbrauch sichtbar.



- Problem: Man merkt meistens nicht, dass der Seitenkanal existiert.

Historie vor der großen Berichterstattung

Technische Universität München

- Juni/ Juli 2017
  - Responsible Disclosure der Bugs von Google an Intel, AMD und ARM
  - Veröffentlichung des KAISER Papers - Eine Lösung für das Meltdown Problem.
  - Öffentlicher Blogpost der die Problematik abstrakt beschreibt, aber noch nicht erfolgreich ausnutzen kann.
- Oktober 2017
  - Intel CEO verkauft im großen Stil Aktienanteile.
- Über Weihnachten: Betriebssystemhersteller arbeiten an Patches.
- Anfang Januar 2018
  - Veröffentlichung eines spekulierenden Blogposts.
  - Klarstellender Blogpost von Google und Veröffentlichung der Schwachstellen:
  - Stellungnahmen verschiedener Hersteller - erste Patches.
- seitdem:
  - Unwissen, fundiertes Halbwissen und hilflose Fixes.
  - Updates werdenherausgegeben und wieder zurück genommen.

Grundlagen

- Bei Spectre und Meltdown handelt es sich auch um Seitenkanalangriffe.
- **Problem:**
  - Prozessoren sind an ihrem physikalischen Performancemaximum.
  - Prozessoren sind sehr komplex.
- Daher: **Verschiedene Techniken um Prozessor Perfomance zu optimieren.**

- Im folgenden soll versucht werden, diese Zusammenhänge grundlegend zu erklären.
  - Paging and (Implicit) Caching,
  - Out-of-Order Execution
  - Speculative Execution
    - Branch Prediction (Branch Trace Buffer)

- **Problem entsteht aus der Kombination der Techniken**

- **Problem:** Speicherzugriffe sind teuer!
  - Die Geschwindigkeit unserer Prozessoren wird in Gigaherz gemessen.
  - In einem Takt kommt ein elektrisches Signal nur über eine gewisse Distanz.
  - Platz ist rar und teuer
- **Daher:** Einführung einer Speicherhierarchie.
- **Beispiel**: Intel i7-4770 (Haswell), 3.4 GHz, 22 nm. RAM: 32 GB.

Typ	Größe	Latenz
L1 Data	32 KB, 64 B/line	4-5 cycles
L1 Instruction	32 KB, 64 B/line	(bei Misspredict) 18-20 cycles
L2	256 KB, 64 B/line	12 cycles
L3	8 MB, 64 B/line	36 cycles
RAM	32 GB	36 cycles + 57 ns
		(0.29 ns / cycle) => ca 230 cycles

Problem: Daten und Programmcode sollen schon bei erster Verwendung im Cache sein!
(Implicit) Caching

- Ein System versucht bald benötigten Speicher vorab in den Cache zu laden.
- **Idee**: Speicherinhalte können „auf gut Glück“ in den Cache geladen werden.
  - Der folgende Programmcode kann direkt in den Cache geladen werden.
  - Für Daten ist separate Analyse der kommenden Instruktionen notwendig.

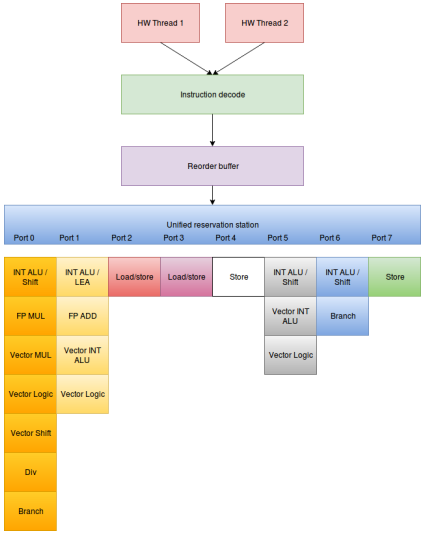
- Findet in den folgenden Instruktionen ein Speicherzugriff statt wird der entsprechende Speicher bereits in den Cache geladen.

Problem: Prozessoren sind komplex

- Prozessoren enthalten spezialisierte Subsysteme.
- Instruktionen werden auf unterschiedlichen Subsystemen ausgeführt.
- z.B.: Arithmetik, Speicherzugriffe, Vektor Logik, ...

Out-of-Order Execution:

- Instruktionen werden parallel ausgeführt.
- Ergebnisse werden gültig gesetzt, sobald alle vorherigen Instruktionen abgearbeitet sind.
  - und keine Fehler aufgetreten sind.
  - z.B. Speicherzugriffsfehler
- Ungültige Zwischenergebnisse werden verworfen.



Problem: Welcher Weg soll bei einer Verzweigung im Programm genommen werden.

Spekulative Ausführung:

- CPU wählt einen Pfad aus und führt diesen spekulativ weiter aus.
  - So lange bis klar ist ob es die richtige Entscheidung war.
- Sollte natürlich seiteneffektfrei sein, da der Reorder-Buffer Instruktionen erst abschließt wenn sie gültig sind.
  - Also wenn klar ist, dass das der richtige Branch war.

Problem: Misspredict penalty, wenn falscher Code spekulativ ausgeführt wird.

- Das Target zu raten ist schlecht.

Besser: Branch Target Buffer loggt für Verzweigungen die Quelle und das Ziel ...

- ... und kann später einen gegebenen indirekten Branch das Ziel vorhersagen.

- Rechteprüfung …
  - … wird beim L1 Cache in der CPU (Reorder Buffer) geprüft.
  - … findet beim Laden erst ab dem L2 Cache statt.
- **Daher:** Daten werden spekulativ unabhängig von den Rechten gelesen.

Speculative Execution verändert den Systemzustand

- Basierend auf spekulativ gelesenen Daten können weitere Berechnungen durchgeführt werden.
  - Mit den Daten kann gerechnet werden.
  - Es können weitere Daten (aus dem RAM) gelesen werden.
- **Resultat**: Diese Daten landen auch im Cache des Prozessors..
  - Wird die spekulative Berechnung verworfen, bleiben die geladenen Daten im Cache.

Abstrakt:

1. Lese Daten aus dem Speicher die nicht gelesen werden dürfen.
2. Lese basierend auf diesen Daten eine andere Stelle im Speicher.
3. Prüfe die Zugriffsgeschwindigkeit der „anderen“ Stelle

```
1  struct array {
2    unsigned long length;
3    unsigned char data[];
4  };
5  struct array *arr1 = ...;
6  unsigned long untrusted_offset = ...;
7  if (untrusted_offset < arr1->length) {
8    unsigned char value =
9       arr1->data[untrusted_offset];
10   ...
11 }
```

- Der Prozessor weiß nicht, dass untrusted_offset größer als arr1->length ist.
- Er wird die nächsten Instruktionen spekulativ ausführen.
- In diesem Beispiel wird der Wert in arr1->data[untrusted_offset] aus dem Speicher gelesen.
  - Allerdings unabhängig davon, ob das Programm den Speicher lesen darf.
- Dies ist kein Problem, da
  - der Prozessor den Zustand später zurückrollen kann, und
  - der Fehler ja schon bei der misspredicition ausgelöst wird.

```
12 struct array {
13   unsigned long length;
14   unsigned char data[];
15 };
16 struct array *arr1 = ...;
17 struct array *arr2 = ...;
18 unsigned long untrusted_offset = ...;
19 if (untrusted_offset < arr1->length) {
20   unsigned char value =
21     arr1->data[untrusted_offset];
22   unsigned long index2 =
23     ((value&1)*0x100)+0x200; ←
24   if (index2 < arr2->length) {
25     unsigned char value2 =
26       arr2->data[index2]; ←
27   }
28 }
```

- Wenn aber bei der Spekulativen Ausführung ...
  - weiterer Speicher gelesen wird,
  - der von der Berechnung abhängt,
  - und der Speicher im Cache verbleibt,
- ... führt dies zu einem Seitenkanal.
- Über Messung des Timings kann nun bestimmt werden ob `value2` $0x200$ oder $0x300$ war.

- Derzeit sind 3 Varianten (Familien) dieses Angriffs bekannt:
  - Variante 1: bounds check bypass (CVE-2017-5753)
  - Variante 2: branch target injection (CVE-2017-5715)

  } Spectre

  - Variante 3: rogue data cache load (CVE-2017-5754)

  } Meltdown

- Grundsätzlich verlaufen alle Varianten relativ ähnlich (in drei Stufen)
  - um jeweils ein Bit zu lesen, wiederhole:
    1. Training des Branch Target Buffers
    2. Spekulative Ausführung des Codes der den Speicher liest
    3. Messen der Cachezugriffszeiten um die Daten aus Schritt 2 zu lesen.

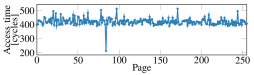


Abbildung: Quelle: Meltdown Paper

Allgemeinste Form:

- **Idee**: Speicher im eigenen Addressraum lesen, der nicht gelesen werden darf.
- Angriff fast wie im vorherigen Beispiel.
- **Ergebnis:** Ein Angreifer kann nicht lesbaren Speicher bitweise lesen.
  - Laut Google ist eine Datenrate von ca 2000 Bytes pro Sekunde möglich.
  - Wenn vom Userspace aus der Kernelspeicher gelesen wird.
- **Problem:** Solcher Code muss im System bereits vorhanden sein, damit der Angriff durchführbar wird.
  - Darum sind hier JIT Compiler von besonderem Interesse.
  - z.B. JavaScript für Exploits aus der Browser Sandbox.
  - oder eBPF (Kernel VM) um Code aus dem Kernel Speicher zu lesen.
- Gegenmaßnahmen Komplex → Backup Slides (auf Anfrage)

- Verhindert werden kann dieser Angriff nur:
  - wenn keine Daten mehr predictive geladen werden,
  - deren Adresse von anderen predictive geladenen Daten abhängt.
- oder:
  - Wenn sichergestellt wird, dass solche Daten nicht im Cache verbleiben,
  - wenn sie nicht vorher schon dort waren.

- Der BTB ist über Prozessgrenzen hinweg gültig,
  - ... verwendet nur ein Subset der Adressbits,
  - ... und kann trainiert werden.
  - Damit kann er von einem anderen Prozess aus beeinflusst werden.
- **Idee:** Trainiere den BTB um anfälligen Code in anderen Prozessen spekulativ auszuführen.
  - Konzeptionell Ähnlichkeit zu ROP, allerdings:
  - Gadget(s) müssen nicht sauber enden (speculative Execution) und
  - nur einen Seitenkanal befüllen.
- Das gelesene Datum wird dann durch das (spekulative) lesen eines Anderen, (geteilten?) Speicherbereiches übertragen.
  - Gemeinsame shared libraries (libc, nt.dll)
- Mit Hilfe von Spectre lässt sich auch der Speicher aus unterschiedlichen Addressräumen und Virtualisierungsebenen lesen.
  - Laut Google ist eine Datenrate ca 1500 Bytes pro Sekunde möglich.

- Meltdown liest Kernelspeicher aus dem Userspace
  - und verwendet dabei den selben Ansatz wie Variante 1.
  - Speicher kann über das Kernel Identity Mapping `0xffffff8800...`[1] gelesen werden.
- Kein Wechsel des Adressraums notwendig, daher kann mit ca 500 kb/s gelesen werden.
- Medial im Fokus, da dieser Angriff einfach aus dem Userspace durchgeführt werden kann.
- Die Technik ist allerdings die gleiche (nur eben ca 300 mal schneller).

[1] unter Linux

Fazit

- Die Angriffe zeigen eine neue Klasse von Seitenkanalangriffen.
- Es ist zu erwarten, dass in der nächsten Zeit weitere Techniken folgen werden.
- Prozessordesign ist zu kompliziert.
  - Derzeit ist ein großer Teil der Funktionalität implementation specific oder unspecified.
  - Es müssen Modelle geschaffen werden um diese Zusammenhänge darzustellen.
    - Ähnlich der Debatten zu: Typesafety, Langsec, ...
- **Problem:** Jetzt werden auf die schnelle Fixes implementiert, deren Auswirkungen nicht bekannt sind / die zu wenig verstanden werden.

Fazit

- Durch den eingeführten Seitenkanal wird die Isolation innerhalb eines Systems aufgeweicht.
- Sobald es für den Angreifer möglich ist Code auf dem Zielsystem auszuführen können Daten extrahiert werden.

- Zwei wesentliche Angriffsszenarien (Angriffsebenen):
  - Betriebssystemisolation
  - Virtualisierung

Fazit

- Betriebssystemisolation
  - Lesen von Daten aus dem Betriebssystemkern oder anderen Prozessen
  - Zum Beispiel Auslesen der Passwortdatenbank des Browsers aus einer Webseite heraus
    - mit Hilfe von JavaScript
  - Lösung: Zugriff auf hochpräzise Timer aus JavaScript heraus deaktiviert.

- Virtualisierung
  - Herauslesen von Daten aus anderen Virtuellen Machinen, auf dem selben physikalischen Host.
  - Möglich, wenn Speicherbereiche zwischen VMs geteilt werden
    - z.B. über Kernel Samepage Merging (KVM), Memory CoW (XEN), ...

Konsequenzen

Und jetzt? Was kann ich tun?

- Immer aktuelle Updates / BIOS Updates einspielen.
  - Microcode Updates werden vom Betriebssystem mitgeliefert.
  - Aber: Updates bringen (unter gewissen Umständen) hohe Perfomanceeinbußen.
- Vorsicht bei der Verarbeitung von Daten in der Cloud
  - Der Cloudanbieter muss sich um die korrekte Isolation kümmern.
- Selbst Informieren um Auswirkungen einschätzen zu können.
- Keine Panik - Aber Vorsicht ist angebracht.
  - Die gilt aber allgemein im Internet!

Warum also der Medienhype?

Technische Universität München

- Neue Angriffsklasse: Seitenkanal
- Viele Geräte betroffen (sowohl Intel als auch Konkurrenten wie AMD oder ARM)

- Angriffsziele: Browser und geteilte Systeme (Cloud Infrastruktur)
  - Es wird absichtlich externer Programmcode ausgeführt.
  - Zum Beispiel JavaScript im Browser.

- Angiff von Außen nicht direkt erkennbar.
  - Ähnlich zu Heartbleed (April 2014)
- Gegenmaßnahmen sind / waren dringend notwendig.

- Retpoline[2] - Vermeidung von Indirect Branches
  - Ersetze Indirect `JMP`s durch `RET`s.
  - Der Returnwert wird beim Aufruf der Funktion auf den Stack geschrieben.
  - Der Stack sollte beim Aufruf des `RET` im Cache liegen.
  - Daher sollte der Prozessor keine spekulative Ausführung starten.
  - Achtung: Im Rahmen der ROP Debatte hatte man `RET` großflächig durch `JMP`s ersetzt.
  - Daher: Lösungsansatz wurde innerhalb von einigen Tagen wieder zurückgezogen.

[2] https://support.google.com/faqs/answer/7625886

- BTB Cache leeren
  - Der Cache des BTB sollte bei jedem Kontextwechsel geleert werden.
  - Ein MSR mit diesem Zweck wird durch die Microcode Updates auf die Prozessoren geliefert.
  - Betriebssysteme nutzen es bei jedem CR3 write bzw. VMEXIT.
- `LFENCE` Instruktion bei Bounds-Checks (Warten, bis alle Speicherzugriffe fertig sind)
- Zugriff auf High Precision Timer verbieten.
- WebKIT: Poisons and Masks[3]
  - Masking: `int tmp = intArray->vector[index & intArray->mask];`
  - Poisoning für Objekt Tests ($\oplus$ mit zufälligem Wert) (siehe Pointer mangling)

[3] https://webkit.org/blog/8048/what-spectre-and-meltdown-mean-for-webkit/

- KPTI / KAISER[4]
  - Eigentlich Schutzmaßnahme für Kernel Address Space Layout Randomization (KASLR)
  - Löst nur zufällig das Meltdown Problem.
  - **Idee**: Kernel Address Space ist nicht mehr in virtuellen Adressraum des Userspace enthalten.
  - Beim Wechsel zwischen Kernel und Userspace wird der TLB geflushed.
  - Damit ist der Kernel Speicher im Userspace nicht mehr zugreifbar.

[4] https://gruss.cc/files/kaiser.pdf

(Microcode-)Updates

- Schritt 1: Einführung von KPTI in Betriebssysteme.
- Schritt 2: Einführung des MSR zum Flush des BTB via Microcode Update
- Schritt 3: Aktivieren der Schutzmechanismen (Beispiel Linux)

(Microcode-)Updates

- KPTI Page Table Isolation (pti)
  - `echo 0 > /sys/kernel/debug/(x86/)pti_enabled`
- Indirect Branch Restricted Speculation (ibrs)
  - Intel: Restricts speculation of indirect branches
  - `echo 1 > /sys/kernel/debug/(x86/)ibrs_enabled` - IBRS im Kernel
  - `echo 2 > /sys/kernel/debug/(x86/)ibrs_enabled` - IBRS im Kernel und im Userspace
- Indirect Branch Prediction Barriers (ibpb)
  - Intel: Ensures that earlier code's behavior does not control later indirect branch predictions.
  - `echo 1 > /sys/kernel/debug/(x86/)ibpb_enabled` - IBPB bei Prozess-/Gastwechsel
  - `echo 2 > /sys/kernel/debug/(x86/)ibpb_enabled` - IBPR bei jedem VMEXIT
- Single Thread Indirect Branch Predictors (STIBP)
  - Intel arbeitet zusätzlich an einem Mechanismus, dass sich die Branch Prediction von zwei Hyperthreads nicht mehr gegenseitig beeinflusst.

IBRS

If IBRS is set, near returns and near indirect jumps/calls will not allow their predicted target address to be controlled by code that executed in a less privileged prediction mode before the IBRS mode was last written with a value of 1 or on another logical processor so long as all RSB entries from the previous less privileged prediction mode are overwritten.