Rapport de recherche

Meltdown & Spectre

Abdourahamane Boinaidi et Aly Nayef Basma

8 novembre 2019

Introduction

Spectre et Meltdown sont les noms donnés à différentes variantes de la même vulnérabilité sous-jacente fondamentale qui affecte presque toutes les puces informatiques fabriquées au cours des 20 dernières années et pourrait, en cas d'exploitation, permettre à des attaquants d'avoir accès à des données précédemment considérées comme complètement protégées.
Les chercheurs en sécurité ont découvert les failles à la fin de 2017 et les ont publiées au début de 2018. Sur le plan technique, il existe trois variantes de la vulnérabilité, chacune ayant son propre numéro CVE[1]; deux de ces variantes sont regroupées sous le nom de Spectre et la troisième est appelée Meltdown.

Meltdown est une nouvelle attaque qui permet de surmonter l'isolation de la mémoire complètement en fournissant un moyen simple pour tout processus utilisateur de lire la totalité de la mémoire du noyau de la machine. Meltdown n'exploite aucune vulnérabilité logicielle, c'est à dire qu'elle fonctionne sur les principaux système d'exploitation. Au lieu de cela, Meltdown exploite les informations des canaux latéraux disponibles sur la plupart des processeurs modernes. Alors que les attaques par canaux auxiliaires[2] nécessitent généralement des connaissances très spécifiques sur l'application cible et sont conçues pour ne divulguer que des informations confidentielles, Meltdown permet à un adversaire qui peut exécuter du code sur le processeur vulnérable d'obtenir un vidage de la totalité du noyau espace d'adressage, y compris toute mémoire physique mappée.

Spectre brise l'isolement entre différentes applications. Il permet à un attaquant de piéger des programmes sans erreur, qui suivent les meilleures pratiques, en révélant leurs secrets. En fait, les vérifications de sécurité de ces meilleures pratiques augmentent en réalité la surface d'attaque et peuvent rendre les applications plus sensibles à Spectre.

Spectre est plus difficile à exploiter que Meltdown, mais il est également plus difficile à atténuer.

---

[1] **Numéro de CVE**: Le programme des vulnérabilités et expositions communes (CVE) répertorie les vulnérabilités des logiciels et des micrologiciels depuis 18 ans.

[2] **Attaque par canal auxiliaire:** une attaque qui recherche et exploite des failles dans l'implémentation, logicielle ou matérielle.

Sommaire

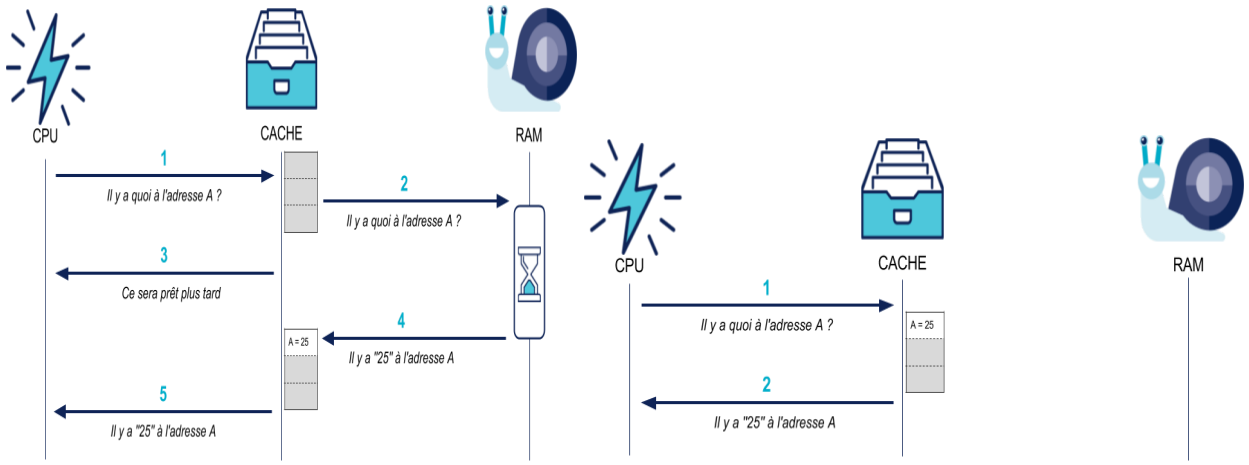
Le Cache et Out-of-order

Il est nécessaire de comprendre le fonctionnement du cache et de l'out-of-order pour mieux comprendre ces deux attaques.

Fonctionnement du cache CPU

La mémoire cache ou antémémoire CPU est une mémoire qui enregistre temporairement des copies de données provenant de la RAM, afin de diminuer le temps d'un accès ultérieur (en lecture) à ces données. Le principe du cache est également utilisable en écriture.

Exemple d'utilisation
En pratique, quand le CPU a besoin d'une valeur en mémoire, il la demande au cache, qui la demande à la RAM.Le cache stocke la valeur récupérée et est capable de la renvoyer plus rapidement au CPU s'il la demande à nouveau :



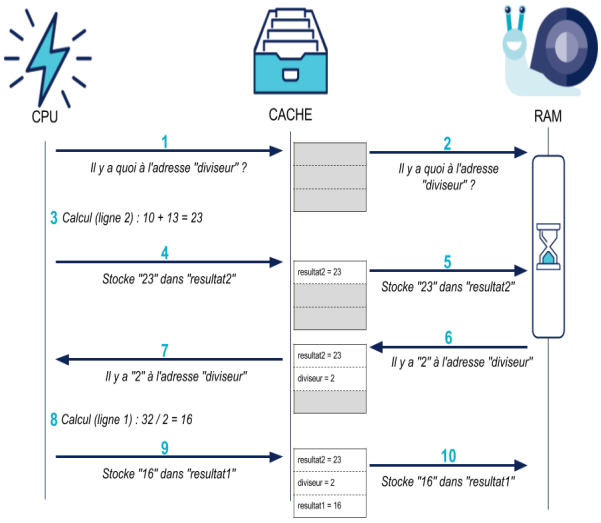
Fonctionnement de l'out-of-order

L'exécution dans le désordre « out of order execution » consiste à réorganiser l'ordre dans lequel les instructions vont s'exécuter dans le processeur. Ces instructions ne sont alors pas forcément exécutées dans l'ordre dans lequel elles apparaissent dans le programme.

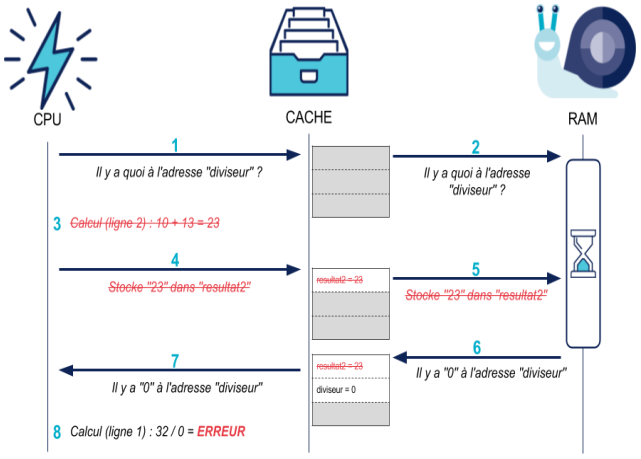
Illustration graphique de deux instructions

1 resultat1 = 32 / diviseur

2 resultat2 = 10 + 13



Exécution sans erreur



Exécution avec erreur

Meltdown

La particularité de Meltdown est que cette attaque exploite une vulnérabilité sur certains processeurs. En effet, les instructions out-of-order peuvent accéder à la mémoire du noyau, alors que cela devrait être interdit. C'est donc en utilisant cette faille que l'attaque permet de récupérer les informations contenues dans l'espace d'adressage du noyau.

L'approche de l'attaque Meltdown se découpe en deux parties :

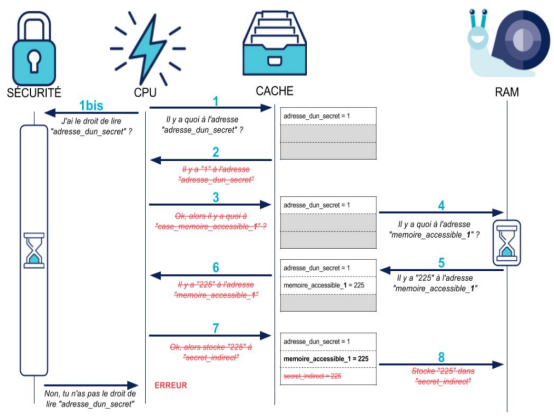
- Etape 1: charger des données dans le cache.
- Etape 2: faire une attaque canaux auxiliaire (Flush and Reload dans notre cas).
- Etape 3: deviner le contenu grâce à l'analyse du cache.

Description de l'attaque

```
1  secret = memoire_noyau[adresse_dun_secret] # ERREUR !

2  si secret est_egal_à 1
3      secret_indirect = memoire_accessible_1
4  sinon
5      secret_indirect = memoire_accessible_0
```

En premier lieu, un attaquant force le processeur à exécuter une séquence d'instructions transitoires qui utilise une valeur secrète inaccessible stockée quelque part dans la mémoire physique (ligne 1). Évidemment, l'accès à la mémoire est interdite, donc une exception va se lever de type SIGSEGV (Segmentation Fault). Cependant, les processeurs effectuent les instructions en parallèle, les instructions suivantes peuvent être exécutées en même temps (ligne2 à 5).



Si secret est égal à 1 : memoire_accessible_1 reste en cache

Jusque là, l'attaquant n'a pas plus d'information car certes, l'exécution out-of-order peut exécuter les lignes 2 à 5 en avance. Pour autant le b mécanisme d'annulation va entrer en jeu, et l'écriture dans secret_indirect sera supprimée. Cependant, memoire_accessible_1 ou memoire_accessible_0 est stocké dans le cache. l'attaquant a donc réussi à charger une donnée dans le cache, la prochaine étape va consister à vider le cache puis refaire l'opération précédente (Flush and Reload). La deuxième étape est indispensable pour la réalisation de la troisième étape. Lorsqu'on vide le cache memoire_accessible_1 et memoire_accessible_0 ne seront plus dans le cache : l'accès aux deux sera lent. Mais, une fois le reste du code passé (ligne 4 à 7), seule l'une des deux cases aura été chargée en cache : son accès sera plus rapide. Il nous suffira juste de vérifier la vitesse d'accès à la mémoire puis afficher les données.

```
1  sortir_du_cache(memoire_accessible_1)
2  sortir_du_cache(memoire_accessible_0)

3  secret = memoire_noyau[adresse_dun_secret] # ERREUR !

4  si secret est_egal_à 1
5      secret_indirect = memoire_accessible_1
6  sinon
7      secret_indirect = memoire_accessible_0

8  si vitesse(memoire_accessible_1) est rapide
9      afficher("le secret vaut 1")
10 si vitesse(memoire_accessible_0) est rapide
11     afficher("le secret vaut 0")
12 sinon
13     afficher("l'attaque a échoué")[2]
```

```
[1] 231
[2] 229
[3] 304
[4] 32  // <--- Temps d'accès le plus rapide,
[5] 274
[6] 299
[7] 257
[8] 311
```

Ainsi l'attaquant peut reproduire ce processus sur les différentes zones mémoire pour lire toutes les données stockées dans la mémoire.

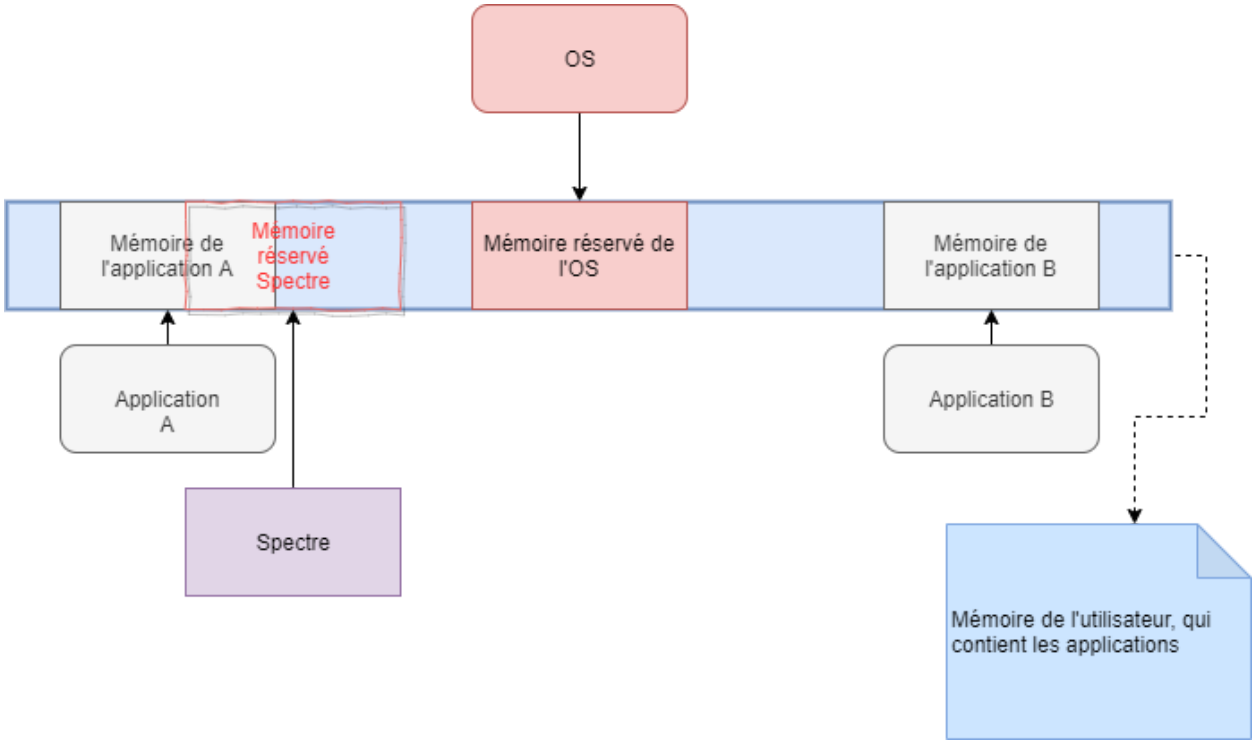
Spectre

Spectre est une vulnérabilité matérielle de certaines implémentations de la prédiction de branchement, qui affecte les microprocesseurs modernes dotés de l'exécution spéculative. Cette vulnérabilité permet de récupérer des informations potentiellement sensibles en forçant un programme à accéder à des zones arbitraires de l'espace mémoire qui lui est allouée.

Comment ça fonctionne ?

Contrairement à Meltdown qui utilise des failles de certains processeurs permettant de lire les adresses du noyau, l'attaque Spectre quant à elle n'utilise pas de faille, mais seulement l'optimisation de prédiction (spéculative) et de cache, pour pouvoir lire n'importe quelle valeur dans le userland d'un processus victime.

L'idée de Spectre est d'entraîner le processeur à suivre un certain chemin lorsqu'une décision doit être prise en utilisant l'optimisation de prédiction, puis de profiter de cette prise de décision entraînée pour que le processeur prenne la branche voulue même si la condition n'est plus respectée.



Exemple 1 - Attaque spectre

L'image du dessus est une exemple qui résume, comment fonctionne l'attaque de spectre.

Sur cette image, on a:

- Une mémoire, qui représente la mémoire de l'utilisateur, et comme on fait une exemple on a quelques zones mémoires réservés.
- Un OS, qui occupe une certaines capacité mémoire de mémoire dans la mémoire.
- Une application A et B qui occupent aussi une partie de la mémoire.
- Une application qui utilise l'attaque spectre, celui-ci s'est superposé sur la mémoire de l'application A, ce qui lui permet d'accéder à l'application A, et d'induire en erreur l'application A pour qu'il écrit dans son espace.

Détail de l'attaque

Pour mieux détailler l'attaque de Meltdown, on va utiliser l'exemple ci-dessous, l'algorithme sera utilisé en se basant sur le langage **C**.

```
if (i < len_array1)
{
    var = array2[array1[i]];
}
```

L'attaquant aura effectué un travail en amont qui aura habitué le processeur au fait que `i` soit inférieur à la longueur du tableau `array1` donc que la condition soit vraie.

Ainsi, à la prochaine exécution, le processeur s'appuyant sur le *Branch Target Buffer* (BTB) se dira que, comme avant, `i` devrait être inférieur à la taille du tableau, donc avant même que la vérification soit faite, il exécutera l'instruction suivante pour gagner du temps, à savoir `var = array2[array1[i]];`. Seulement cette fois, l'attaquant a décidé d'utiliser un `i` arbitraire, qu'il contrôle, et qui est supérieur à la taille du tableau `array1`.

La conséquence est que l'instruction `var = array2[array1[i]]` sera tout de même exécutée en prédiction, donc que `array1[i]` sera évalué, et vaudra par exemple 12. Une fois cette valeur trouvée, 12 dans l'exemple, l'index 12 du tableau array2 va être lu, et le contenu sera assigné à var.

Bien entendu, le processeur va ensuite se rendre compte que le i fourni n'était pas valide, donc il va annuler les instructions qu'il a pré-exécutées, donc var n'aura finalement pas de valeur.

Cependant, la zone mémoire correspondant à `array2[12]` (avec `12 = array1[i]`) aura tout de même été mise en cache, laissant une trace.

Une fois cette mise en cache effectuée, de la même manière que pour Meltdown, la technique du **flush + reload** est utilisée pour voir quel index de `array2` a été mis en cache, permettant de découvrir la valeur secrète `array[i]` avec i étant trop grand normalement pour passer le test initial.

Pour généraliser et trouver la valeur à n'importe quelle adresse, nous savons que `array1[i]` est équivalent à `*(array1 + i)` ou `*(&array1[0] + i)` en **C**. Donc si l'attaquant veut voir ce qui se passe en mémoire à l'adresse `0xbfff1234` par exemple, un bête calcul permet de trouver le `i` qu'il doit fournir

```
// Nous voulons ceci
&(array1[i]) == 0xbfff1234;

// Or
&(array1[i]) == (&array1[0] + i) == (array1 + i);

// Donc
i = 0xbfff1234 - array1;
```

Une fois ce calcul en tête, l'attaquant peut extraire n'importe quel octet de la mémoire du programme en cours, dont des zones cachées contenant des mots de passe, des clés de cryptographie ou autres secrets.

Conclusion

Ces deux attaques font beaucoup parler d'elles car il n'y a pas de manière évidente et simple de les patcher donc de s'en protéger.

Concernant Meltdown, il est par exemple possible de changer totalement le mode de gestion de la mémoire en faisant en sorte que le noyau et le processus aient deux espaces d'adressage distincts. Ainsi, nous pouvons passer du mode de gauche (partage de l'espace d'adressage) au mode de droite (séparation des espaces d'adressage)

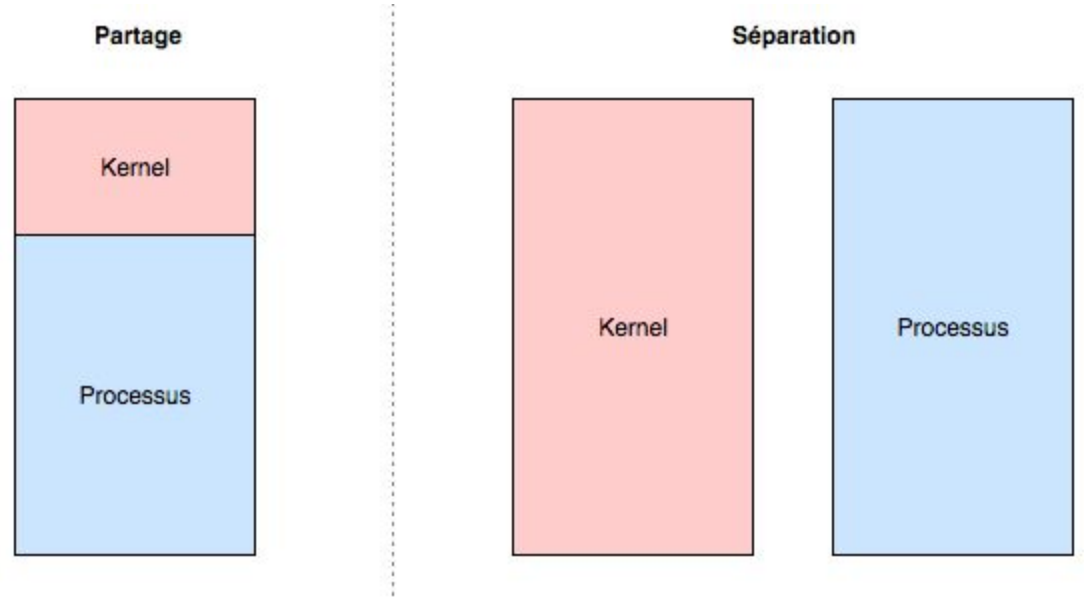


Schéma - Séparation noyau, processus

Evidemment, le processeur devra souvent changer de contexte d'exécution (user - kernel), et devra alors mettre en cache beaucoup d'informations puisqu'il devra alterner entre deux espaces d'adressage indépendants, d'où les baisses de performances dont on parle.

En ce qui concerne Spectre, il est beaucoup plus compliqué de trouver un moyen de s'en protéger. En effet, l'attaque utilise des éléments intrinsèques à l'architecture de l'ordinateur, sans utiliser de vulnérabilité, et sans accéder à des zones mémoire interdites par le matériel. Il faut donc revoir en profondeur le fonctionnement des optimisations.

Pour ce qui de Meltdown il faut s'assurer que les patches les plus récents sont installés sur votre système, malheureusement pour spectre, il faudra attendre des nouveaux processeurs.

Références

https://github.com/IAIK/meltdown

https://www.nbs-system.com/blog/attaque-informatique-faille-meltdown-exploitation/

https://fr.wikipedia.org/wiki/Ex%C3%A9cution_dans_le_d%C3%A9sordre

http://cryptofails.com/post/70097430253/crypto-noobs-2-side-channel-attacks

https://beta.hackndo.com/meltdown-spectre/

https://meltdownattack.com/

https://blog.octo.com/quand-votre-cpu-parle-trop-la-faille-meltdown-et-ses-consequences/

https://www.lemonde.fr/blog/binaire/2018/01/05/lattaque-meltdown/

https://www.kyos.ch/spectre-meltdown-analyse/

https://www.lebigdata.fr/explications-attaques-meltdown-spectre