L'attaque MELTDOWN

David Monniaux

L’attaque Meltdown

5 janvier 2018, blog *Binaire*, https://www.lemonde.fr/blog/binaire/2018/01/05/lattaque-meltdown/

par : David Monniaux, directeur de recherche au CNRS, laboratoire Verimag

Les réseaux sociaux et les blogs spécialisés en sécurité bruissaient de rumeurs depuis une semaine (pourquoi des modifications si urgentes dans le système de gestion de mémoire du noyau Linux, alors que d’habitude il faut des mois et des mois pour que le moindre changement soit accepté ?). Comme d’habitude lorsque des trous de sécurité majeurs sont découverts, ceux-ci n’étaient documentés que sous embargo, c’est-à-dire qu’on a d’abord informé les industriels ou groupes de développeurs susceptibles de fournir des correctifs, en leur laissant un délai suffisant, avant de publier les articles décrivant les problèmes.

Il y a en fait deux catégories d’attaques publiées ce jour : MELTDOWN et SPECTRE, qui partagent certaines caractéristiques. J’ai publié un autre billet sur SPECTRE, dont je vais reprendre ici quelques éléments explicatifs. Je vais discuter ici de MELTDOWN, en me basant sur la lecture de l’article décrivant les attaques (Lipp et al., *Meltdown*). Les attaques MELTDOWN sont celles contre lesquelles Microsoft, Apple et les développeurs de Linux ont intégré des contre-mesures. Je vais tenter de les expliquer à un niveau ne nécessitant pas de connaissances particulières en informatique.

Dans un ordinateur, un ou plusieurs processeurs exécutent des séquences d’instructions de calcul (additions, soustractions, multiplications, lecture ou écriture de données dans la mémoire). Ce sont ces instructions qui constituent les logiciels : quelle que soit la complexité ou le domaine d’application de celui-ci, ou le langage de programmation utilisé, on en revient toujours à l’exécution d’une suite de petites instructions comme cela.

On décrit parfois l’exécution de ces instructions de la façon suivante : le processeur lit l’instruction dans la mémoire, la décode (s’agit-il d’une addition, d’une soustraction, etc.), récupère éventuellement dans la mémoire les données dont elle a besoin, exécute l’opération demandée, puis écrit éventuellement son résultat dans la mémoire. C’est ainsi, en effet, que fonctionnaient les processeurs du début des années 1980 (Motorola 68000, par exemple).

Ce mode de fonctionnement est inefficace : il faut attendre que chaque étape soit achevée pour aborder la suite. On a donc fait par la suite des processeurs qui, bien qu’ils semblent, du point de vue du programmeur, exécuter successivement les instructions, les exécutent en fait comme sur une chaîne d’assemblage automobile (on parle, en terme techniques, de pipeline) : une unité du processeur décode, dès que l’instruction est décodée on la transfère aux unités qui lisent en mémoire qui la gèrent tandis que l’instruction suivante est décodée et que l’opération de calcul de l’instruction précédente est exécutée. On en est même venu à avoir des processeurs qui réordonnent l’exécution de parties d’instruction afin d’utiliser au maximum leurs unités, voire des processeurs qui tentent d’exécuter deux programmes à la fois sur les mêmes unités en tirant parti du fait que certaines sont inoccupées (*hyperthreading*) ! Ce qu’il faut retenir, c’est qu’il y a de nos jours, dans les processeurs à haute performance (dont ceux des PC portables, de bureau ou serveurs), des

mécanismes extrêmement complexes qui essayent, grosso modo, de simuler une exécution « comme en 1980 » alors que ce n'est pas ce qui se passe dans la machine. Voyons certains de ces mécanismes.

J'ai dit plus haut qu'il fallait souvent chercher dans la mémoire de la machine (la RAM) les données nécessaires à l'exécution d'une instruction. Or, l'accès à la RAM prend du temps, beaucoup plus que l'exécution d'une instruction : cet écart entre la vitesse d'exécution des instructions et le temps nécessaire pour obtenir une donnée de la RAM a crû au cours du temps. Pour compenser, on a intégré dans les processeurs des mécanismes de *mémoire cache* qui, grosso modo (c'est en réalité bien plus compliqué) retiennent dans le processeur les données accédées les plus récemment et évitent le trajet vers la mémoire si la donnée recherchée est dans le cache.

Par ailleurs, les processeurs intègrent des mécanismes de protection de la mémoire, qui isolent les applications les unes des autres et isolent le système d'exploitation des applications. Ils évitent qu'un banal bug dans, par exemple, un traitement de textes, provoque des dysfonctionnements dans le système d'exploitation, avec possibilité de panne plus large que l'application défectueuse et pertes de données. Le déclenchement de ces mécanismes dans une application utilisateur se manifeste généralement par la fermeture autoritaire de celle-ci par le système d'exploitation et l'affichage d'un message d'erreur (« faute générale de protection » sous Windows, « erreur de segmentation » sous Linux…).

Ces mécanismes de protection de la mémoire servent également à la sécurité, puisqu'ils permettent d'isoler les uns des autres les utilisateurs de la machine. Ceci est bien évidemment d'une extrême importance quand cette machine fait cohabiter des utilisateurs très divers, par exemple un serveur utilisé par les étudiants d'une université, encore plus lorsqu'il s'agit d'un serveur de *cloud computing*, c'est-à-dire d'une machine située chez un prestataire (OVH, Microsoft Azure, Amazon Web Services…) où des clients sans aucun rapport les uns avec les autres louent la possibilité de lancer des applications.

L'attaque MELTDOWN est possible parce que dans certains cas, sur certains modèles de processeurs, notamment du fabricant Intel, le processeur commence à exécuter des instructions dépendant d'un accès mémoire illégal avant de se rendre compte de l'illégalité de celui-ci. Bien entendu, il ne s'agit pas d'une erreur aussi naïve que d'autoriser la récupération directe d'une donnée qui devrait rester protégée : lorsque le processeur s'aperçoit de l'erreur de protection, il rétracte l'opération illégale ainsi que celles qui dépendaient de lui. Or, et c'est le même phénomène qui est exploité dans les attaques SPECTRE, cette rétractation est imparfaite : si des instructions qui ont commencé d'être exécutées avant d'être rétractées ont provoqué des chargements dans le cache, les données ainsi chargées dans le cache y restent. Il est ensuite possible d'y détecter leur présence par des moyens indirects (la présence ou l'absence de certaines données dans le cache produit des différences de temps d'exécution qu'il est possible de mesurer). J'ignore pourquoi ces processeurs ne commencent pas par vérifier la protection avant de lancer les chargements dans le cache. L'article dit que c'est peut-être pour des raisons d'efficacité.

Quelles conséquences de l'attaque MELTDOWN ? Dans certaines circonstances (cela dépend des mécanismes précis de protection), un simple utilisateur peut lire le reste de la mémoire de la machine, y compris la mémoire du noyau et celle des autres utilisateurs, accédant à toutes les données y compris sensibles (mots de passe). C'est donc une attaque particulièrement gênante pour

les prestataires de *cloud computing* (mais, dans leur cas, sa possibilité dépend du mécanisme technique d'isolation des utilisateurs les uns des autres, car tous ne sont pas vulnérables), ou pour les gestionnaires d'environnements avec un grand nombre d'utilisateurs ayant le droit de lancer des logiciels de leur choix (universités, entreprises). Dans le cas de machines personnelles, elle ne permet qu'une *escalade de privilège* : l'individu malveillant doit déjà trouver un moyen de lancer un logiciel de son choix sur cette machine, et ensuite seulement peut exploiter la faille pour obtenir l'accès à des données auxquels ce logiciel ne devrait pas déjà avoir normalement accès ; cela me semble un problème plus limité.

Comment pallier l'attaque MELTDOWN ? Microsoft (Windows), Apple et les développeurs de Linux ont des solutions. Bien évidemment, seule celle de Linux est documentée : elle consiste à « cacher » totalement la mémoire du système d'exploitation aux applications, au lieu de, comme actuellement, l'exposer mais avec l'indication « cette mémoire est protégée et n'est accessible que par le système d'exploitation ». Cela implique qu'à chaque fois que le système d'exploitation sera appelé par l'application, celui-ci change la « carte mémoire » utilisée, et devra la changer à nouveau dans l'autre sens au retour dans l'application. Ceci a bien entendu un coût, les applications faisant beaucoup appel au système d'exploitation (on parle de 30 % d'efficacité en moins pour des bases de données) étant plus fortement touchées que celles qui font juste des calculs en mémoire. Une rectification du matériel serait bien entendu préférable, mais implique probablement de changer le parc informatique existant.