4MMCSR - Projet Rowhammer

Amaury Boin
amaury.boin@phelma.grenoble-inp.fr

Charles d'Hondt
charles.dhondt@gmail.com

Keywords

DRAM; securité; rowhammer

Introduction

Dans les années 90, IBM a mené des recherches sur l'impact des rayons cosmiques sur les cellules constituant les mémoires. Leurs chercheurs ont mesuré 1 erreur induite par mois pour 256Mo de RAM. Les rayons cosmiques peuvent donc bien changer la valuer d'un bit mais la probabilité qu'un tel phénomène se produise est assez minime.

En 2013, une équipe de chercheurs de la Carnegie Mellon Univeristy met en évidence une faille de sécurité se basant sur un phénomène similaire. En effet, avec la miniaturisation de la mémoire, un modèle de DRAM (Dynamic Random Access Memory) est constitué de cellules tellement petites que la consultation de l'une de ses cellules produit une interaction avec les cellules voisines. En consultant en boucle une cellule, ces interactions peuvent avoir le même effet qu'un rayon cosmique et la valeur du bit voisin se retrouve changé. En partant de ce constat, une attaque de très bas niveau peut être mis en place.

1. THÉORIE

1.1 Changement des bits dans la DRAM

L'isolation de la mémoire est une propriété importante d'un système informatique, un accès mémoire doit en effet ne pas impacter les cellules mémoires qui l'entoure. Cependant, la montée en puissance de la technologie a permis d'augmenter la densité des cases mémoires. Et a permis aux cases mémoires de la DRAM de s'influencer électroniquement les unes avec les autres. Ici, l'attaque rowhammer se base sur ce principe : qu'un accès mémoire sur une case mémoire de DRAM permet de changer les bits des cases mémoires qui l'entourent et donc de corrompre leurs mémoires. Ces erreurs de petubations peuvent être observés en réalisant des accès mémoire fréquents sur une même case mémoire. Des chercheurs de la Carnegie Mellon Univeristy ont en effet montrés qu'il suffit de 139 000 accès mémoire sur une cellule mémoire de DRAM pour induire une erreur de bit sur une cellule voisine. De plus, en moyenne une cellule mémoire sur 1700 est susceptible d'être affectée et de changer de bit.

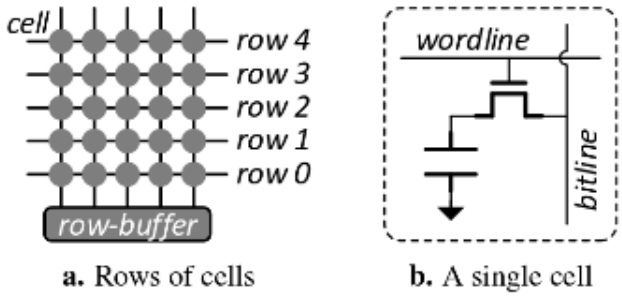


Figure 1. DRAM consists of cells

Figure 1: Organisation d'une cellule de DRAM

1.1.1 Organisation d'une cellule de DRAM

La mémoire DRAM est une matrice de cellule mémoire de DRAM. Chaque cellule de DRAM est composée d'une capacité et d'un transistor. En fonction de si la capactité est chargée ou déchargée, cela représente la valeur du bit. Chargée signifiant que le bit est à 1 et déchargée que le bit est à 0.

Chaque cellule est à l'intersection de deux fils perpendiculaires : la "wordline" horizontale et la "bitline" verticale. La "wordline" connecte toutes les cellules d'une ligne tandis que la "bitline" connecte toutes les cellules de la colonne. Quand on veut mettre les bits mémoires d'une ligne dans le buffer, on met la "wordline" à un voltage positif ce qui permet d'actionner les transistors qui actionennent eux même à leur tour les capacités qui se connectent à leurs "bitline". Cela permet de transférer la mémoire de la ligne dans le buffer.

1.1.2 Accès mémoire d'une cellule de DRAM

L'accès à une case mémoire de DRAM se fait d'une façon précise qui consiste en trois étapes, dans un premier temps il faut ourvrir une ligne mémoire, cela consiste à mettre la "wordline" à un voltage positif et donc à transférer ses bits donc ses données dans le buffer. Puis, on accède au buffer en lisant ou écrivant sur une des ses colonnes. Et finalement on ferme la ligne mémoire en vidant le buffer. Il faut fermer la ligne mémoire avant de pouvoir en ouvrir une autre.

De plus le processeur qui contrôle la mémoire ne permet

pas de réaliser des accès mémoires à la fréquence que l'on souhaite. On est limité par un retard qui est le temps minimum que l'on doit attendre quand on souhaite réalise de nombreux accès mémoires. L'ordre typique de ce retard est de 50 nanosecondes et donc limite la fréquence d'accès mémoire à une fois toutes les 50 nanosecondes.

1.1.3 *Erreurs de pertubations de la DRAM*

Les erreurs de pertubations qui sont à la base de l'attaque rowhammer arrivent quand il y a une interaction physique entre deux composants électriques qui auraient du être isolés électroniquement. En particulier, ici on s'interessera à une erreur de pertubation lié à la "wordline". En effet si on met la "wordline" à un voltage positif de nombreuses fois. On voit que certaines cellules ont des fuites électriques et perdent leurs charges bien plus rapidement que d'habitude, permettant un changement de bits.

En effet, en temps normal une cellule mémoire doit être actualisée au moins une fois toutes les 64ms. Cela est dû à divers mécanismes de fuites de charges électriques liés à la couche physique de la technologie. Dans les DRAM, le temps maximum qu'une cellule mémoire doit passer sans être rechargée est de 64 ms. Cependant l'erreur de pertubation induit des pertes de données dues à des fuites électriques en un temps bien plus court, la cellule mémoire de la DRAM perd donc la valeur de son bit.

Il existe trois hypothèses sur la façon dont la "wordline" intéragit électronniquement sur les cases mémoires environnantes. Dans un premier temps il est possibleque le fait de mettre la "wordline" à un voltage positif induise du bruit sur une "wordline" adjacente à cause de couplages électromagnétiques. Une deuxième hypthèse serait la formation de ponts entre des fils ou des capacités de différentes cellules mémoires, chose assez commune dans le cadre de cette technologie. La troisième hypothèse serait qu'utiliser la "wordline" pendant des centaines d'heures l'abimerait de façon permanante par "injection de porteurs chauds".

1.1.4 *Conséquences des erreurs de pertubations*

Les erreurs de pertubations ne vérifient pas deux règles primordiales que la mémoire doit vérifier. D'une part, une lecture ne doit pas modifier la mémoire à n'importe quelle adresse et d'autre part une écriture ne doit modifier le contenu des bits que à l'adresse à laquelle il se trouve. En pratique, pour provoquer l'attaque rowhammer et les erreurs de pertubations, il suffit d'ouvrir et fermer une ligne mémoire, ainsi tant la lecture que l'écriture permet de réaliser ces erreurs et l'attaque. Un changement de bits (et donc du contenu de la mémoire) peut évidemment être exploité afin d'accéder à des parties de la machine non autorisés. Des programmes malicieux pourraient être crées afin qu'à partir d'accès mémoire fréquents qui demandent peu de droits sur une machine permettent de toucher à des zones qui demandent d'avoir plus d'authorisation.

1.1.5 *Susceptibilité des DRAM*

En réalisant de nombreux tests sur différents modules de mémoires de trois différents constructeurs, on arrive à définir et vérifier des propriétes des erreurs de perturbations.

En effet, dans un premier temps, le nombre d'erreurs dépend énormément du constructeur et de la date de frabication. Au fil des ans, plus la densité des cellules mémoires augmente plus le nombre d'erreurs est important. De plus, les DRAM des trois constructeurs sont tous susceptibles à cette attaque.

Puis, on vérifie bien que l'intervalle d'actualisation de la mémoire (Partie 1.1.3) affecte grandement le nombres d'erreurs. En effet, plus la durée de cet intervalle est court, plus le nombre d'erreurs est faible car la cellule n'a pas le temps d'avoir des fuites éléctriques et a le temps pour moins d'accès mémoire entre deux actualisations. En revenche, quand cette durée augmente, on voit que le nombre d'erreurs augmente pour les mêmes raisons.

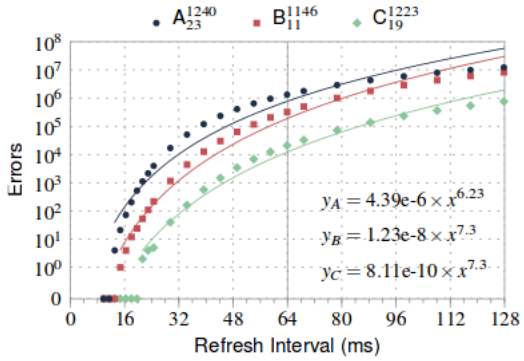


Figure 2: Nombre d'erreurs en fonction de la période de rafraîchissement

Et enfin que l'intervalle d'activation est d'autant plus faible que le nombre d'erreurs est grand. En effet si la fréquence d'activation est élevé, il y a plus de chance d'avoir une erreur car il y aura plus d'accès mémoire dans le même lapse de temps.

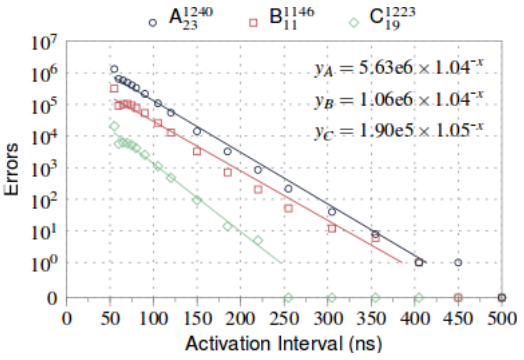


Figure 3: Nombre d'erreurs en fonction de l'intervalle d'activation

1.2 Exploitation de la faille

1.2.1 *Explication*

On peut tenter une attaque sur JVM (Java Virtual Machine) en utilisant le rowhammer. Le point de départ de cette attaque est une faille dans le système de type de Java. En effet, quand deux pointeurs de types distincts pointent la même adresse, on peut alors contourner le système de type. Cela nous permet d'écrire une fonction qui lit et écrit des endroits de mémoire arbitraires de l'espace du programme et par conséquent exécute du code arbitraire.

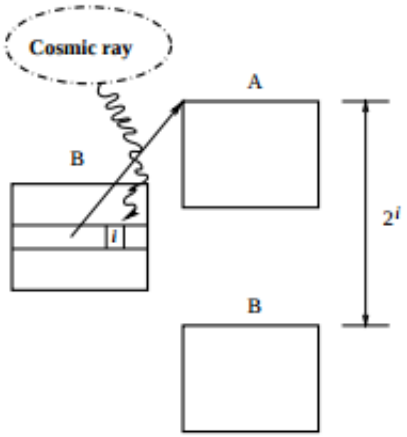
1.2.2 *Comment faire pointer deux pointeurs de types incompatibles sur une même adresse ?*

Définissons deux classes A et B de telle sorte que la taille de ses instances, avec leur en-tête soit une puissance de 2 :

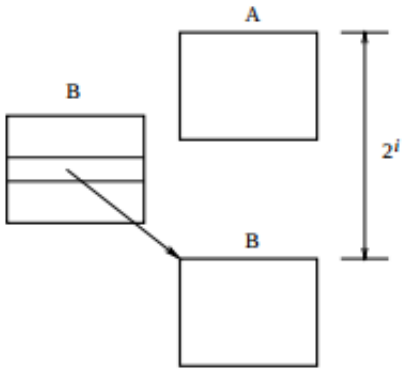
```
class A {
 A a1;
 A a2;
 B b;
 A a4;
 A a5;
 int i;
 A a7;
};
```

```
class B {
  A a1;
  A a2;
  A a3;
  A a4;
  A a5;
  A a6;
  A a7;
};
```

Si un rayon cosmique venait à changer le i-ème bit d'un des mots de B :



On obtiendrait alors ceci :



Et on aurait un pointeur de type A pointant sur B. Cette étape pourrait être forcée en appliquant ce que l'on sait du rowhammer et en modifiant activement les bits voisins du bit i nous intéressant.

Par ailleurs, lors du déréférencement, la JVM prend donc notre objet de type B pour un objet de type A. Puis, à l'aide de ce code :

```
A p;
B q;
int offset = 6 * 4;
void write(int address, int value) {
   p.i = address - offset ;
   q.a6.i = value ;
}
```

Un appel de write(a,v) écrit v à l'adresse offset + (a - offset), c'est-à-dire a. On peut donc écrire à des endroits arbitraires. Ceci constitue ensuite la porte d'entrÃl'e pour qu'un logiciel malicieux puisse mener une attaque.

1.3 Contremesures

Pour contrer une attaque Rowhammer, plusieurs approches sont possibles.

- Tout d'abord, on peut régler le problème au niveau de la puce elle-même. En effet, si l'effet Rowhammer était pris en compte par les concepteurs de puce, cela pourrait être une solution. Cependant, les cellules mémoire étant de plus en plus petites, elles deviendront d'autant plus vulnérables.
- On peut également mettre en place des codes correcteurs d'erreurs (ECC). Cette méthode permet effectivement de régler efficacement les attaques sur un bit. Mais installer des ECC est coûteux : en effet, sur une architecture 64 bits, cela reviendrait à ajouter 8 bits supplémentaires pour chaque bus de 64 bits. Or, perdre 12,5 % de la mémoire n'est pas viable économiquement pour les constructeurs de mémoire aujourd'hui. Par ailleurs, les ECC peuvent déceler les erreurs sur 1 ou 2 bits mais sont inefficaces face aux attaques sur plusieurs bits. Cependant, comme le Rowhammer joue sur des probabilités, si l'on subit une attaque sur plusieurs bits, il y a de fortes chances que l'on détecte des erreurs sur 1 ou 2 bits. Ainsi, si on référence les erreurs détectées, les ECC, même si elles ne peuvent pas détecter les erreurs sur plus de deux bits, elle peuvent mettre en garde d'une éventuelle attaque.
- Comme nous l'avons vu, en diminuant la période de rafraîchissement à 8.2ms, le Rowhammer ne se manifeste plus dans nos sytèmes actuels. Passer de 64ms à 8.2ms comme période de rafraîchissement aurait des conséquences lourdes sur les performances. Plutôt que de consacrer de 1,4 à 4,5 % de leur temps au rafraîchissement de la mémoire, les modules dépenseraient 11 à 35% de leur temps.
- En identifiant les cellules victimes, on peut les recycler en cellules de réserve. Cette opération de recherche exhaustive peut prendre plusieurs jours mais cette étape

peut être faite lors de la manufacture mais également par l'utilisateur final. Cependant, cette approche demande de garder en mémoire et d'identifier les bus déficients ce qui aura un impact sur les performances du système.

- Une autre solution serait de rafraîchir les bus adjacents aux bus "chauds" (comprendre les bus consultés fréquemment). Seulement, compter le nombre d'accès est très coûteux et il est difficile de placer la limite entre bus très fréquentés normalement et bus subissant une attaque. Quand un bus "chaud" est détecté, cela mène à de nombreux rafraîchissements. Ainsi, cette solution, bien que fonctionnelle pour contrer les attaques rowhammer, n'est pas encore très efficace.

- Enfin, une méthode efficace qui a été concue dans le but de contrer l'effet Rowhammer dans les DRAM est l'Activation Probabiliste des Bus Adjacents (en anglais, PARA). Le principe est simple : quand un bus est consulté, il y a une probabilité très faible p (avec $p << 1$) que l'un des bus voisins soit rafraîchi. Par conséquent, plus un bus est consulté fréquemment, plus il y a de chances que ses voisins soit rafraîchi. Comme il s'agit d'une approche probabiliste, on ne peut pas prévenir les attaques Rowhammer avec certitude. Néanmoins, les résultats sont plus que probants : si l'on fait 100 000 lancers de pièce entre deux rafraîchissements et que l'on fixe $p = 0.001$, alors, la probabilité d'obtenir une erreur en un an est de $9,4 \times 10^{-14}$, ce qui est plus que négligeable.
  Par ailleurs, cette méthode nécessite peu de ressources. En effet, à aucun moment on ne garde en mémoire un quelconque état. L'impact sur les performances du PARA est donc très minime et permet de réduire les erreurs à une probabilité acceptable.

-

2. EXPERIENCES

Attaque sur des systèmes Intel et AMD

On peut réaliser une attaque sur des systèmes Intel et AMD grâce au code suivant (1a).

```
1 code1a:
2   mov (X), %eax
3   mov (Y), %ebx
4   clflush (X)
5   clflush (Y)
6   mfence
7   jmp code1a
```

Figure 4: Code 1a

Cela permet de générer de nombreuses requêtes de lecture qui vont permettre d'induire l'attaque rowhammer par changements de nombreux bits, en effet on réalise une lecture à chaque fois qu'on accède à la mémoire. Ce code permet d'accéder à deux lignes de la DRAM alternativement et donc force le contrôle de la mémoire dans le processeur à ouvrir et fermer ces deux lignes mémoires de façon alternative. On itère ensuite ce code des millions de fois et cela pour différentes paire d'adresses jusqu'à que toute la mémoire soit ouverte et fermée des millions de fois.

Cette éxécution induit des milliers d'erreurs de bits dans la DRAM. Cela varie en fonction de l'architecture du processeur de la machine ciblée. La tableau suivant indique le nombre de bits qui ont changé en fonction de l'architecture du processeur pour les cas de figure où tous les bits étaient à 1 initialement et celui où ils étaient tous à 0.

Bit-Flip	Sandy Bridge	Ivy Bridge
'0' → '1'	7,992	10,273
'1' → '0'	8,125	10,449

Figure 5: Nombre de changements de bits

Similairement, on a testé un code qui ne touche qu'à une seul adresse mémoire au lieu de la paire d'adresses du code 1a.

```
1 code1b:
2   mov (X), %eax
3   clflush (X)
4
5
6   mfence
7   jmp code1b
```

Figure 6: Code 1b

Ce code semble avoir le même but et exécuté des millions de fois devrait induire des changements de bits sur les adresses proches de celle lue. En réalité, le contrôleur de mémoire ouvre seulement une fois la ligne mémoire, la lis autant de fois que le code est éxécuté et la ferme une seule fois. Comme les changements de bits sont induits par l'ouverture et la fermeture d'une ligne. Ici on n'observe aucune erreur de pertubation, c'est à dire de changemnent de bits, avec l'éxécution de ce code. Le tableau ci-dessous résume ce concept :

Access Pattern	Disturbance Errors?
1. $(open–read–close)^N$	**Yes**
2. $(open–write–close)^N$	**Yes**
3. $open–read^N–close$	No
4. $open–write^N–close$	No

Figure 7: Erreurs de pertubation en fonction des méthodes d'accès

Conclusion

Nous avons vu que l'attaque rowhammer consistant à provoquer des erreurs de pertubations sur des lignes de DRAM en

accédant de multiples fois aux lignes adjacantes de DRAM. Cette attaque est très peu documentée et est possible sur de nombreux systèmes et machines. Nous avons vu comment la provoquer et la caractériser et avons proposé de nombreuses contre-mesures pour faire face à cette faille de sécurité.

Références

- https://www.cs.princeton.edu/ appel/papers/memerr.pdf
- https://github.com/CMU-SAFARI/rowhammer/blob/master/kim-isca14.pdf