

TP ROW HAMMER

Sécurité

Membres :

Bonneau Baptiste
Centeno Matéo
Ferreira Pierre
Jean Mathilde
Zborowski Lucas

Contexte

Vous faites partie d'une équipe de hackers d'élite engagée par une organisation secrète. Votre mission, si vous l'acceptez, consiste à exploiter la faille de sécurité Row Hammer sur un serveur appartenant à une entreprise rivale, connue sous le nom de "asaN". La asaN détient des informations confidentielles qui pourraient compromettre la sécurité nationale, et votre organisation a été chargée de les récupérer.

Vous avez 30 minutes pour infiltrer le serveur de la asaN en utilisant l'attaque Rowhammer, tout en évitant de vous faire détecter. Soyez conscients que vous agissez dans une zone juridiquement grise et que les conséquences de votre mission peuvent être graves si vous êtes pris.

Rappel

Pour rappel, l'attaque Row Hammer est une vulnérabilité matérielle qui se produit lorsque des opérations de lecture et d'écriture fréquentes et rapides sont exécutées sur des lignes adjacentes de cellules de mémoire (ou "rows") dans une puce de RAM. Cette activité intensive peut provoquer des interférences électromagnétiques qui altèrent les données stockées dans les cellules de mémoire adjacentes, permettant ainsi à un attaquant de corrompre ou de manipuler des données en exploitant cette faiblesse.

Partie 1: Mise en bouche

1) Memory Game

a) Vous avez à disposition un programme d'initialisation aux hackers, [HowRHWork.py](#), les règles sont plutôt simples, vous devez trouver à quelle ligne il y a eu un bit flippé. Pour cela, vous avez 4 options :

- 1 : Afficher l'état actuel de la mémoire
- 2 : Marteler : À l'aide d'un puissant coup de marteau, tentez de faire changer la valeur d'un bit en mémoire. Attention, ce processus ne marchera pas à tous les coups.
- 3 : Submit : soumettez votre réponse, il s'agit du seul moyen de vaincre l'exercice, vous devez entrer la ligne à laquelle le bit a changé de valeur.
- 4 : Retour : Abandon total de l'attaque.

Ce cours programme n'est qu'une simulation de RowHammer et n'est en rien dangereux pour le pc, il s'agit d'un exercice cours, pour comprendre les bases de l'attaque de Rowhammer.

P.S : Il vous sera probablement nécessaire d'installer la librairie "colorama" de python pour faciliter la complétion de cet exercice.

b) Question bonus : Dans les logs de la asaN, il semblerait que la ligne numéro 964 a été précédemment modifiée, où est-ce que le coup de marteau ayant réussi à accomplir un bit flipping a pu être effectué ? cette ligne pourrait être intéressante à exploiter...

2) Le code Corrompu

Sur le dépôt Code#0, une source fiable vous a fourni un code non complet, ce code est capable de simuler une attaque RowHammer. Malheureusement, ce code a été corrompu, et seuls certains des commentaires de documentations pourront vous permettre de remplir ces trous. Vous retrouverez ce document dans `~/Top_Secret/CorruptedCode`. Il ne sera pas forcément utile de tester votre code jusqu'à l'apparition d'un message "Oui, un bit a flipé en mémoire." car ce processus peut être beaucoup trop long, vous n'aurez uniquement besoin de compléter les trous du codes représentés par `/***/`. Bonne chance, vous en aurez besoin.

Partie 2 : Test de Rowhammer par Google

Vous venez de découvrir que votre fidèle allié Google a mis à disposition un code permettant de créer une attaque Row Hammer. Étant donné que votre code n'est pas super performant, lancez une attaque en utilisant ce nouveau programme.

A réaliser sur VDN

- 1) Par magie, vous avez eu accès à un ordinateur et vous devez récupérer des informations sur l'ordinateur. Recherchez des informations sur la RAM utilisée sur cet ordinateur. *Votre intuition révèle l'existence de la commande dmidecode.*
- 2) Votre allié vous a fourni le code source présent au repository suivant :
<https://github.com/google/rowhammer-test.git>
- 3) Démarrez l'attaque avec `./make.sh` puis `./rowhammer_test`
Qu'observez-vous ?
Il vous faudra sûrement plusieurs heures avant de trouver une faille avec ce programme ! Il est temps d'essayer autre chose ! Éteignez-le.

- 4) Votre programme s'est arrêté !

Que s'est-il passé ?

Qu'elle est l'adresse obtenue et qu'elle est celle que nous devons avoir par défaut ?

```
Iteration 5098 (after 5139.67s)
Took 91.6 ms per address set
Took 0.915664 sec in total for 10 address sets
Took 21.196 nanosec per memory access (for 43200000 memory accesses)
This gives 377431 accesses per address per 64 ms refresh period
Checking for bit flips took 0.084910 sec
Iteration 5099 (after 5140.67s)
Took 89.9 ms per address set
Took 0.898797 sec in total for 10 address sets
Took 20.805 nanosec per memory access (for 43200000 memory accesses)
This gives 384513 accesses per address per 64 ms refresh period
Checking for bit flips took 0.084819 sec
Iteration 5100 (after 5141.66s)
Took 91.9 ms per address set
Took 0.918996 sec in total for 10 address sets
Took 21.273 nanosec per memory access (for 43200000 memory accesses)
This gives 376062 accesses per address per 64 ms refresh period
error at 0x7fcea84ccb8: got 0xffffffffffffefff
Checking for bit flips took 0.086122 sec
** exited with status 256 (0x100)
```

Exercice Bonus (trop compliqué pour ce tp :)

01101000 01110100 01110100 01110000 01110011 00111010 00101111 00101111

[... exo 1 ...]

01100111 01101100 01100101 00101110 01100011 01101111 01101101 00101111 01100100
01101111 01100011 01110101 01101101 01100101 01101110 01110100 00101111 01100100
00101111 00110001 01001111 00110000 00110110 01010101 [3] 01011111 01100111
01010110 01101101 00110111 01011001 01010000 01001100 00110101 01100100
01010101 00111000 01001101 01101000 01110110 00110101 TygGwPLI 01100111
00110111 00110100 01000011 00110110 01000010 01010101 00101111 01100101

01100100 01101001 01110100 00111111 01110101 01110011 01110000 00111101 01110011
01101000 01100001 01110010 01101001 [2]