Детектування атак типу RowHummer за допомогою частотних масивів

Detecting RowHummer attacks using frequency arrays

Валентин Мазурок
Corresponding author: аспірант кафедри кібербезпеки фізико-технічного інституту, e-mail: valentin.mazurok@gmail.com, ORCID: 0009-0006-2174-0800

Володимир Луценко
кандидат технічних наук, старший науковий співробітник НАН України, доцент кафедри кібербезпеки фізико-технічного інституту, e-mail: lutsenkovn@ukr.net, ORCID: 0000-0001-7632-1730

Київськи політехнічний інститут імені Ігоря Сікорського, м. Київ, Україна

Valentyn Mazurok
Corresponding author: Postgraduate student of Department of Cybersecurity, Institute of Physics and Technology, e-mail: valentin.mazurok@gmail.com, ORCID: 0009-0006-2174-0800

Volodymyr Lutsenko
кандидат технічних наук, старший науковий співробітник НАН України, доцент кафедри кібербезпеки фізико-технічного інституту, e-mail: lutsenkovn@ukr.net, ORCID: 0000-0001-7632-1730

Igor Sikorsky Kyiv Polytechnic Institute, Kyiv, Ukraine

Received: February 20, 2025 | Revised: February 26, 2025 | Accepted: February 28, 2025

DOI: 10.33445/sds.2025.15.1.12

Мета роботи: створення алгоритму захисту оперативної пам'яті DRAM від вразливості RowHammer, що не має вразливості нерівномірного оновлення.

Результати дослідження: показано механізм детектування RowHammer на основі частотних масивів, що не має вразливості нерівномірного оновлення.

Практична цінність дослідження: Показаний алгоритм захисту може покращувати захищеність даних в інформаціних системах та сприяти кращому розумінню та покращенню методів захисту пам'яті DRAM вцілому

Цінність дослідження: Представлено новий інструментарій детектування атак типу RowHammer на основі частоти доступу. Продемонстровано результати роботи на нових видах пам'яті DDR5.

Майбутні дослідження Це дослідження відкриває шляхи для майбутніх досліджень динаміки розвитку захисту від атак типу RowHummer та суміжних атак на пам'ять сторонніми каналами.

Тип статті: практичний.

Purpose: creating an algorithm to protect DRAM from the RowHammer vulnerability that does not have the uneven refresh vulnerability.

Findings: demonstrated a RowHammer detection mechanism based on frequency arrays that is not vulnerable to uneven updating vulnerability.

Practical implications: presented protection algorithm can improve data security in information systems and contribute to a better understanding and improvement of DRAM memory protection methods in general.

Value: presented new tools for detecting RowHammer attacks based on access frequency. Demonstrated protection results on new types of DDR5 memory.

Future research: this research opens new ways for future research into the development of defenses against RowHummer-type attacks and related memory attacks via side channels.

Papertype: practical.

Ключові слова: кібербезпека, RowHammer, DRAM, атаки на пам'ять.

Key words: cybersecurity, RowHammer, DRAM, memory attacks.

Вступ

Ізоляція пам'яті є фундаментальною характеристикою безпечної та надійної обчислювальної системи. Звернення до певної адреси в пам'яті не повинно впливати на дані, що зберігаються в інших комірках. Проте зі зменшенням технологічного процесу мікросхеми пам'яті стають дедалі вразливішими до електромагнітних наведень і взаємного впливу між комірками.

Дослідження, представлене на ISCA 2014 [1], демонструє механізм виникнення помилок у стандартних мікросхемах динамічної оперативної пам'яті (DRAM), які використовуються в сучасних системах. Було виявлено, що багаторазове звернення до однієї й тієї самої адреси може спричинити спотворення даних у сусідніх комірках. Зокрема, якщо рядок DRAM неодноразово відкривається (активується) і закривається (попередньо заряджається), то після певної кількості таких операцій, виконаних протягом одного інтервалу оновлення DRAM, один або кілька бітів у фізично суміжних рядках можуть змінитися. Це явище отримало назву RowHammer [2, 3].

Теоретичні основи дослідження

Помилки спотворення даних виникають, коли між двома компонентами схеми (наприклад, конденсаторами, транзисторами або провідними доріжками), які мають бути ізольовані, виникає надмірно сильна взаємодія. Залежно від того, які саме елементи взаємодіють і яким чином, можуть виникати різні типи спотворень.

Один із таких специфічних механізмів впливає на стандартні чіпи DRAM від трьох провідних виробників. Якщо напруга в певному рядку даних змінюється багаторазово, у сусідніх рядках з'являються вразливі комірки, в яких заряд починає витікати значно швидше. Якщо такі комірки активуються занадто часто, вони не здатні утримувати достатній рівень заряду навіть протягом 64 мс — стандартного часу оновлення DRAM. Як наслідок, дані в цих комірках пошкоджуються, що призводить до накопичення помилок. Саме цей механізм і відомий під назвою RowHammer.

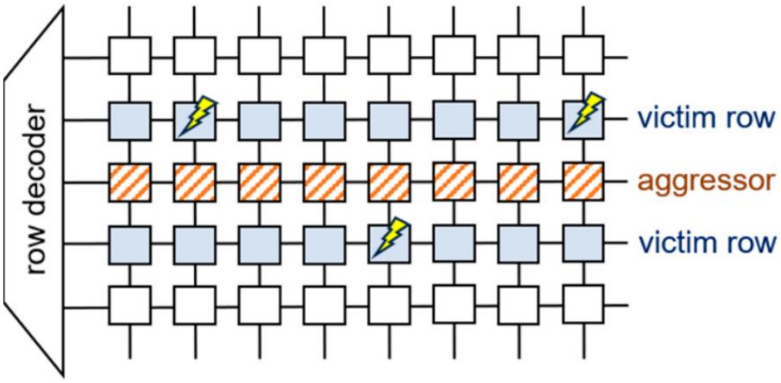

Рисунок 1 – Схема атаки RowHummer

Лічильникові механізми захисту від RowHammer наразі забезпечують найнадійніший рівень захисту. Вони використовують лічильники активацій рядків для ідентифікації рядків-агресорів. На відміну від імовірнісних підходів, такі методи можуть гарантувати запобігання бітовим помилкам, оскільки агресор завжди буде виявлений до того, як зможе пошкодити критичні дані.

Оскільки рядки DRAM періодично оновлюються, усі лічильники в таких методах також необхідно регулярно скидати. Проте оновлення всіх рядків DRAM не відбувається одночасно, тоді як скидання лічильників зазвичай здійснюється синхронно. Оновлення рядків розподіляється невеликими інтервалами часу в періоді оновлення t_{REF}. Однак контролер пам'яті не має інформації про те, який саме рядок оновлюється в поточний момент, що унеможливлює синхронізацію скидання лічильника з оновленням відповідного рядка. Як наслідок, більшість сучасних методів виявлення та захисту на основі лічильників виконують одночасне скидання всіх лічильників на початку періоду оновлення. Це створює певні ризики для ефективності контрзаходів.

Через те, що більшість лічильників скидається незалежно від оновлення рядків, атакуючий може скористатися цим, спрямовуючи атаку на ті рядки, чиї лічильники будуть скинуті посеред атаки до їх оновлення. Це може дозволити атаці залишитися непоміченою. На рисунку 2 показано приклад атаки, в якій оновлення рядка не синхронізоване зі скиданням лічильників. У такому випадку, коли лічильники обнуляються, фактична кількість активацій (ACT) рядка перевищує його поточний лічильник. Якщо цей рядок виступає агресором і перевищує порогове значення (A_{RH}), його лічильник все одно буде меншим за реальну кількість активацій, оскільки він був скинутий під час атаки. У такому сценарії атака не фіксується захисним механізмом.

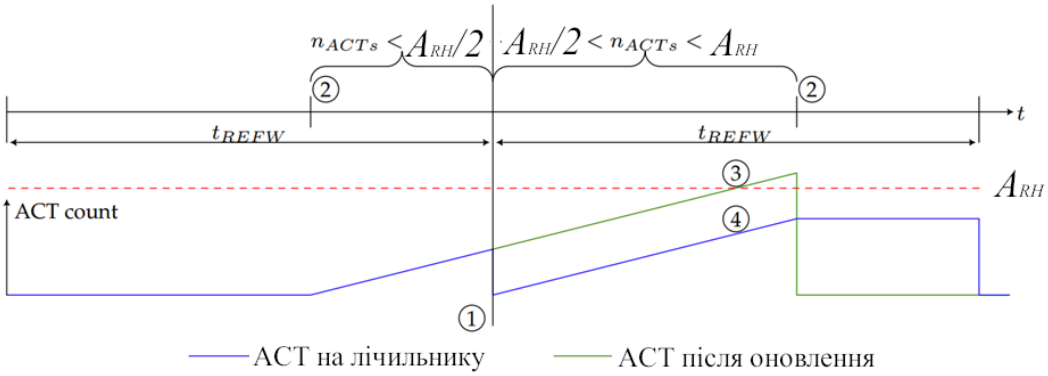

Рисунок 2 – Еволюція лічильника доступу при несинхронному оновленні

Для вирішення цієї проблеми різні механізми реалізують різні підходи. Наприклад, BlockHammer [4] застосовує подвоєний механізм і чергує скидання лічильників, тоді як Graphene [5] розділяє поріг виявлення на два, що змушує використовувати вдвічі більше лічильників порівняно з початковою конфігурацією. Це звісно відображається як на продуктивності так і на енерговикористанні.

Постановка проблеми

Проблема безпеки пам’яті типу RowHammer стала важливою в сучасних комп’ютерних системах, оскільки такі атаки можуть призвести до витоку конфіденційної інформації. Лічильники доступу є надійним інструментом виявлення доступу до пам’яті, але попередні дослідження не змогли імплементувати його в повній мірі для запобігання новим типам атак. Через це стоїть технічне завдання імплементації нових типів детектування атак RowHammer, що не має недоліків нерівномірного оновлення та не використовує надмірно ресурси системи для захисту.

Методологія дослідження

Запропонований механізм виявлення на основі частотного масиву поєднує використання лічильників активацій із оцінкою їх частоти. Для спрощення розгляду ми припускаємо, що виконання двох команд активації (ACT) завжди вимагає фіксованого часу, навіть якщо між ними видається періодична команда оновлення (REF). Іншими словами, ми аналізуємо лише часовий інтервал, упродовж якого можуть виконуватися команди активації пам’яті.

Нехай у межах одного банку пам’яті протягом періоду оновлення може бути здійснено N_C команд активації. Враховуючи порогове значення збурення T_{RH} та те, що два сусідні рядки жертви можуть зазнати впливу з одного рядка “агресора”, мінімальна необхідна кількість активацій (ACT) для одного рядка, що призводить до зміни бітів, становить:

$$A_{RH} = \frac{T_{RH}}{2}$$

Тоді кількість рядків використаних в атаці можна знайти як:

$$R_A = \frac{N_C}{A_{RH}}$$

Тому середній період між двома ACT на агресора не може перевищувати $P = R_A \cdot t_{RC}$ для успішної атаки. Тут t_{RC} – час на одну команду активації. Іншими словами, якщо середній період ACT рядка перевищує P, він не є агресором, оскільки не може завершити атаку протягом вікна оновлення (t_{REFW}). Щоб відстежувати лише потенційного агресора, ми можемо відстежувати лише рядки, які активуються досить часто, щоб бути агресорами. Відстежувані рядки будуть збережені в таблиці. Записи таблиці складаються з пари ключ-значення, де ключ — це

ідентифікатор рядка, а значення — час видалення з таблиці. Прості атаки можна виявити, використовуючи лише цей лічильник: коли його значення наближається до порогу виявлення (≈ A_{RH}), рядок можна класифікувати як агресорний і запобігти збуренню даних, застосовуючи відповідні контрзаходи, наприклад, оновлюючи сусідні рядки або обмежуючи доступ до пам'яті для атакуючого процесу.

Після того як рядок позначено як агресорний, його запис у системі можна очистити, щоб звільнити місце для нових записів. Такий підхід ефективний проти атак RowHammer, які регулярно виконують команди активації (ACT) для своїх агресорних рядків. У цьому випадку запис у таблиці зберігається, а лічильник збільшується з кожною активацією, поки не досягне порогу виявлення. Однак цей підхід має дві ключові проблеми, які необхідно додатково вирішити. По-перше, атакувальний процес може утримувати запис у масиві протягом тривалого часу, постійно виконуючи команди ACT, але не досягаючи порогового значення. Це дозволяє агресору послідовно блокувати записи один за одним, що зрештою може призвести до необхідності використання великої кількості лічильників і механізмів моніторингу, щоб запобігти повному заповненню таблиці записів.

Щоб вирішити цю проблему, можна змінити метод оцінки агресорних рядків. Замість того, щоб покладатися лише на значення лічильника, слід враховувати частоту активацій. Формула для оцінки потенціалу рядка як агресора в атаці RowHammer f_{RH} розраховується з урахуванням таких параметрів: C – значення лічильника, t_{expt} – час видалення запису, t – поточний час та P – максимальна затримка між активаціями (ACT) для агресорних рядків.

$$f_{RH} = \mathrm{C} \times \frac{t_{exp} - t}{P}$$

Зміну цього значення відносно лічильника проілюстрована на рисунку 3 Поріг для f_{RH} при якому розглядаємо ряд як агресора $f_{RH} = A_{RH}$. Якщо рядку агресора видається ACT через регулярний інтервал $\Delta t \in [2t_{RC}; P)$, значення f_{RH} можна обчислити після ACT за формулою:

$$f_{RH} = c\frac{t_0 + cP - (t_0 + (c-1)\Delta t)}{P} = c\frac{c(P - \Delta t) + \Delta t)}{P}$$

де t_0 це час першої команди активації, виданої рядку. Якщо атака видає ACT рядку якомога повільніше ($\Delta t \to P$), f_{RH} досягає A_{RH} для $c = c_{макс} \approx A_{RH}$. І навпаки, якщо атака видає ACT якнайшвидше ($\Delta t \to 0$), f_{RH} досягає порогу A_{RH}. для $c = c_{мін} \approx \sqrt{A_{RH}}$. Точне значення c_{min} можна обчислити як найменше ціле значення C що задовольняє нерівність $f_{RH} \geq A_{RH}$, з $\Delta t = \Delta t_{мін} = 2 \times t_{RC}$.

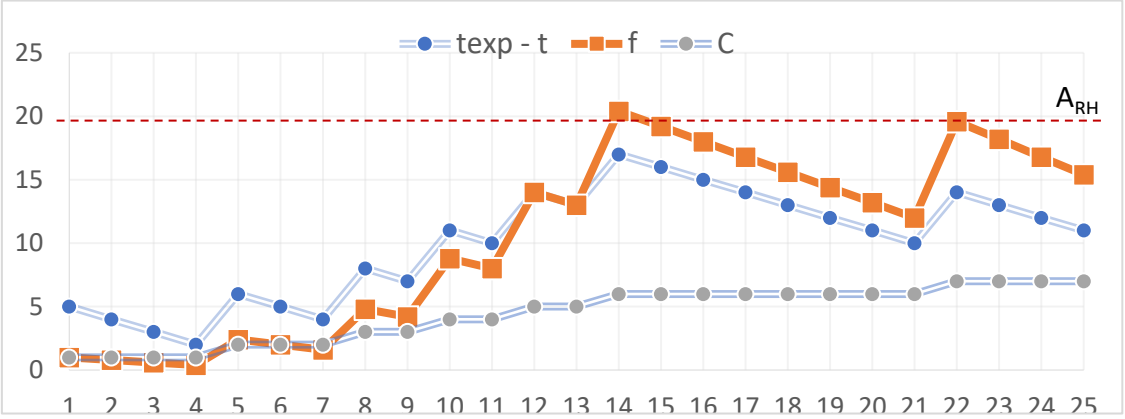

Рисунок 3 – Зміна значень лічильника часу та частоти в таблиці доступу при атаці RowHammer

Результати

Для роботи механізм реалізує таблицю з двома адресованими вказівниками на пам'ять Content Addressable Memory (CAM) та одним масивом. Перший CAM CAM_R містить ідентифікатори рядків, другий CAM_T – час видалення, а масив CNT містить значення лічильників. Причина використання CAM для ідентифікаторів рядків і часу видалення полягає в тому, що обидва вони потребують пошуку, коли видається ACT. CAM_R використовується для пошуку, якщо в активованому рядку вже є запис, а CAM_T використовується для пошуку записів для заміни або видалення. Однак значення лічильника в CNT використовуються лише для розрахунку f_{RH}. Запис у механізмі виявлення має однаковий індекс для всіх масивів.

З точки зору алгоритму, у таблиці реалізовано такі функції:

CAM_P реалізує функції пошук(*search(рядок)*), встановлення(*set(індекс*, *рядок*)).

CAM_T реалізує функції пошуку, видалення і встановлення по часу *searchExpired*(*t*), searchAvailable(*t*), get(*індекс*), set(*індекс*, *t*).

Додаткова функція видалення (*delete*(*індекс*)), який видаляє запис з усіх трьох компонентів таблиці.

Обчислення мінімальної кількості записів можна здійснити шляхом імітації атаки, яка намагатиметься використати якомога більше записів.

```
1   read t_ACT
2   read W
3   read d_min
4   read HCf irst
5   P = W*t_ACT / (A_RH)
6   repeat
7   c_min = findCmin(P, d_min)
8   P = t_ACT * (W-d_min*(c_min-2)) / A_RH-c_min+1 m
9   until c_min is fixed
10  n_entries = d_min
11  t_stop = (c_min - 1) * (P - d_min * t_ACT ) + 1
12  while t_stop > P + 1 do
13      n_ACT = (t_stop-d_min*t_ACT -1) / P
14      t_stop = min(n_ACT * (P - d_min * t_ACT ) + 1, t_stop - n_ACT * d_min * t_ACT )
15      n_entries = n_entries + d_min
16  n_entries = n_entries + t_stop/t_ACT
```

Рисунок 4 – Імітація атаки з максимальною кількістю записів

Розглядаючи ж типову імплементацію для захисту на основі лічильників можна обрати вже досліджену DDR4 [5]. Використовуємо сталі параметри синхронізації, для такого типу пам’яті та знайдений поріг A_{RH} = 16384 [6]. Для впровадження на рівні банку відповідні параметри часу є такими: N_C = 1.33 · 106, d_{min} = 2.

Для простоти виберемо t_{ATC} = 1, тобто, один тік за ACT. Алгоритм показаний на рисунку 4 дає нам значення P = 83, c_{min} = 130 і $n_{записи}$ = 447. Зверніть увагу, що в цій конфігурації значення P і c_{min} достатньо малі, щоб їх можна було визначити лише за одну ітерацію. Максимальне значення, якого може досягнути лічильник, c_{max} = 16189

Зі значеннями P і c_{min}, ми можемо порахувати $(t_{exp} - t)_{макс}$ = 10452, і також розмір таймера для різних значень $n_{циклів}$. Для $n_{циклів}$ = 2, таймер повинен утримувати 2 · $(t_{exp} - t)_{макс}$ = 20904, що вимагає принаймні 15 біт. Для $n_{циклів}$ = 3, таймер мав би утримувати лише до 1.5 · $(t_{exp} - t)_{макс}$ = 15678, що вимагає 14 біт. Розглядаючи типовий DDR4 з 2^{16} = 65536 рядів, для $n_{циклів}$ = 2, кожен запис таблиці складатиметься з 16 (ідентифікатор рядка) + 15 (час до видалення) + 14 (лічильник ACT) = 45біти. Таблиця має $n_{записи}$ = 447 записів, загальний розмір таблиці становитиме 447 · 45 = 19.6 *Kib* (включаючи як CAM, так і масив CNT). Знайдені значення представлено в таблиці 1.

Таблиця 1 – параметри алгоритму для різних типів пам'яті

DDR тип	A_{RH}, тис	t_{REFW}, мс	N_C, млн	t_{ACT}	Лічильники	Розмір масивів CAM_P+CAM_T	Розмір масиву CNT
DDR3	69,2	64	1,25	1	114	3.35 Kib	1.91 Kib
DDR4	16	64	1,33	1/12	465	12.2 Kib	6.41 Kib
DDR5	4,8	32	0,661	1/28	701	17.6 Kib	8.25 Kib

При цьому в усіх випадках рівень детектування склав 99.7%, переважаючи підхід захисту іншими методами на основі лічильників. Для порівняння, реалізація Graphene на рівні банку для розглянутої DDR4 буде використовувати відповідно CAM 4,8 Кіб, а BlockHammer для тієї ж конфігурації використовуватиме масив лічильників 26 Кіб. Тобто така реалізація не тільки не матиме недоліків з нерівномірним оновленням лічильників, а і займатиме значно менше місця 31.5 % у першому і 24.5 % у другому випадку відповідно.

Висновки

Проблема безпеки пам'яті, зокрема атаки RowHammer, стала критичною загрозою для сучасних комп'ютерних систем, оскільки може призвести до витоку конфіденційної інформації. Лічильники доступу є ефективним засобом для моніторингу операцій з пам'яттю, проте попередні дослідження не змогли повністю інтегрувати їх для запобігання новим варіаціям атак.

Запропонований механізм детектування атак, що базується на частотному масиві, продемонстрував 99,7% ефективності у виявленні RowHammer. Цей метод виявився більш оптимальним у використанні пам'яті порівняно з існуючими рішеннями, такими як Graphene і BlockHammer, скорочуючи витрати пам'яті на лічильники на 31% та 24,5% відповідно. Це забезпечує високий рівень захисту, однак його основним недоліком є необхідність попереднього встановлення порогових значень, що може обмежувати ефективність методу при зміні характеристик нових модулів пам'яті.

Фінансування

Це дослідження не отримало конкретної фінансової підтримки.

Конкуруючі інтереси

Автори заявляють, що у них немає конкуруючих інтересів.

Список використаних джерел

1. Kim Y. et al. (2014), "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," in ISCA (pp. 361-372). https://doi.org/10.1109/ISCA.2014.6853210.
2. Aga M. T. et al. (2014), "When Good Protections go Bad: Exploiting anti-DoS Measures to Accelerate Rowhammer Attacks," in HOST (pp. 8-13), https://doi.org/10.1109/HST.2017.7951730.
3. Chandrasekar K. et al. (2016), "Exploiting Expendable Process-margins in DRAMs for Run-time Performance Optimization", in DATE (pp. 1-6), https://doi.org/10.7873/DATE.2014.186.
4. A Giray Yağlikçi et al. Blockhammer: Preventing RowHammer at low cost byblacklisting rapidly-accessed DRAM rows. In HPCA, 2021.(pp. 345-358), https://doi.org/10.1109/HPCA51647.2021.00037.
5. Yeonhong Park et al. Graphene: Strong yet lightweight row hammer protection.In MICRO, 2020. (pp. 1-13), https://doi.org/10.1109/MICRO50266.2020.00014.

6. Al-Ars Z.et al. (2006), “DRAM-Specific Space of Memory Tests,” in ITC, https://doi.org/10.1109/TEST.2006.297701.

References

1. Kim Y. et al. (2014), “Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors,” in ISCA (pp. 361-372). https://doi.org/10.1109/ISCA.2014.6853210.
2. Aga M. T. et al. (2014), “When Good Protections go Bad: Exploiting anti-DoS Measures to Accelerate Rowhammer Attacks,” in HOST (pp. 8-13), https://doi.org/10.1109/HST.2017.7951730.
3. Chandrasekar K. et al. (2016), “Exploiting Expendable Process-margins in DRAMs for Run-time Performance Optimization”, in DATE (pp. 1-6), https://doi.org/10.7873/DATE.2014.186.
4. A Giray Ya˘glikçi et al. Blockhammer: Preventing RowHammer at low cost byblacklisting rapidly-accessed DRAM rows. In HPCA, 2021.(pp. 345-358), https://doi.org/10.1109/HPCA51647.2021.00037.
5. Yeonhong Park et al. Graphene: Strong yet lightweight row hammer protection.In MICRO, 2020. (pp. 1-13), https://doi.org/10.1109/MICRO50266.2020.00014.
6. Al-Ars Z.et al. (2006), “DRAM-Specific Space of Memory Tests,” in ITC, https://doi.org/10.1109/TEST.2006.297701.