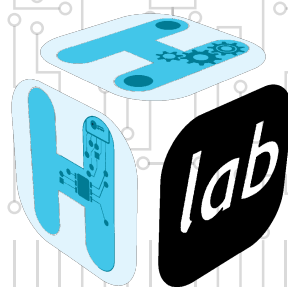

Attaques microarchitecturales : concepts et analyse



Analyse technique H²Lab



Informations

Créée en 2020, H2Lab est une association loi 1901 dont l'objectif est de concevoir, développer et diffuser des solutions open-hardware et open-source dédiées à l'expérimentation autour des systèmes embarqués.

Ses principaux axes de travail sont :

- Conception de plateformes matérielles pour l'analyse hardware et software
- Développement d'alternatives ouvertes aux outils propriétaires, souvent opaques
- Publication de guides techniques et de recommandations pour promouvoir les bonnes pratiques en matière de sécurité, de confidentialité et de durabilité dans l'écosystème des objets connectés et des systèmes embarqués

Soutien aux chercheurs, enseignants et étudiants via la mise à disposition d'outils, de contenus pédagogiques et de formations.

Partenariats académiques pour encourager la mutualisation des ressources et des savoirs.

Historique des révisions

Version	Date	Auteur	Modifications
1.0	Octobre 2026	H2Lab	Version initiale du guide

Table des matières

Informations	i
1 Glossaire et acronymes	1
1.1 Glossaire	1
1.2 Liste des acronymes	1
2 Introduction et périmètre	4
2.1 Objectif du guide	4
2.2 Public visé	4
2.3 Périmètre et terminologie	4
2.4 Résumé exécutif	5
3 Fondamentaux microarchitecturaux	6
3.1 Architecture visible et implémentation réelle	6
3.1.1 Structures internes utiles à l'analyse	6
3.2 Exécution hors ordre et spéculation	7
3.2.1 Pourquoi la retraite ordonnée ne suffit pas	7
3.3 Canaux auxiliaires usuels	8
3.3.1 Deux familles de mesure du cache	8
3.4 Chaîne générale d'une attaque transitoire	9
3.4.1 Ce qui distingue une fuite d'une compromission complète	10
4 Panorama des attaques publiées	11
4.1 Repères chronologiques	11
4.2 Spectre : exécution spéculative contrôlée	12
4.2.1 Spectre v1 – contournement spéculatif de vérification	12
4.2.2 Spectre v2 – injection de cible de branchement	13
4.2.3 Autres variantes Spectre	14
4.3 Meltdown et fuites liées aux fautes	15
4.3.1 Meltdown (2018)	15
4.3.2 Foreshadow / L1TF et failles voisines	16
4.4 Échantillonnage de données internes : MDS et dérivés	16
4.5 Injection de valeur et exécution spéculative par gadgets	17
4.6 Rowhammer et perturbations DRAM	18
4.6.1 Principe et premières démonstrations	18
4.6.2 Contourner ou dépasser les défenses	19
4.6.3 Mesures adaptées aux produits embarqués	20
4.7 CLKSCREW : fautes provoquées par la gestion énergétique	20
4.7.1 Pourquoi cette attaque entre dans le guide	20

4.7.2	Rappel sur DVFS et marges temporelles	20
4.7.3	Principe de CLKSCREW	21
4.7.4	Impacts démontrés	21
4.7.5	Mesures de protection et limites	21
4.8	Canaux temporels hors exécution transitoire	22
5	Impacts sur un système embarqué	23
5.1	Lecture transversale des familles d'attaque	23
5.2	Frontières de sécurité concernées	23
5.3	Conséquences et facteurs de risque	23
5.4	Particularités de l'embarqué	25
5.4.1	Cœurs asymétriques et SoC hétérogènes	25
5.4.2	Temps réel et systèmes à ressources contraintes	26
5.4.3	DMA, accélérateurs et périphériques	26
5.4.4	Démarrage, mise à jour et durée de vie	26
6	Contre-mesures et méthode d'analyse	27
6.1	Défense en profondeur	27
6.1.1	Silicium et microcode	27
6.1.2	Noyau, hyperviseur et firmware	28
6.1.3	Compilateur et application	28
6.1.4	Mémoire et DRAM	29
6.2	Évolutions et limites des mitigations	29
6.2.1	Comment analyser un contournement annoncé	29
6.3	Procédure de qualification pour un produit	30
6.4	Questions de revue de conception	30
7	Conclusion	32
	Licence	36



1

Glossaire et acronymes

1.1 Glossaire

Exécution spéculative Exécution anticipée d'instructions avant que le processeur ait déterminé qu'elles appartiennent au chemin effectivement retenu. Les résultats architecturaux d'un chemin abandonné sont annulés, mais certains effets microarchitecturaux peuvent subsister

État architectural État visible selon l'architecture du processeur : registres, mémoire, exceptions et flot de contrôle tels que définis par l'ISA

État microarchitectural État interne qui conditionne le coût des opérations : caches, prédicteurs, files d'attente, TLB, buffers de chargement et structures de DRAM

Canal auxiliaire Propriété observable, distincte de la sortie fonctionnelle prévue, dont la variation permet d'inférer une information comme le temps d'exécution, l'occupation du cache ou la consommation

Exécution transitoire Instructions exécutées avant la résolution d'une dépendance ou d'une prédiction, puis retirées du chemin architectural lorsque la prédiction est infirmée

Résident privilégié Logiciel exécuté avec des privilèges élevés comme le noyau, l'hyperviseur, le firmware ou le moniteur sécurisé

Rowhammer Perturbation électrique de cellules DRAM provoquée par l'activation répétée de lignes voisines, susceptible d'inverser des bits dans des lignes non ciblées

Barrière de spéculation Instruction ou séquence qui empêche, selon l'architecture concernée, certaines instructions suivantes d'être exécutées avant la résolution d'un événement antérieur

1.2 Liste des acronymes

ABI Application Binary Interface, interface binaire entre composants compilés

AES Advanced Encryption Standard, algorithme de chiffrement symétrique

AMD Advanced Micro Devices, fabricant de processeurs

ARM Famille d'architectures de processeurs Arm

BHI Branch History Injection, injection dans l'historique de branchement

BIOS Basic Input/Output System, micrologiciel d'initialisation de la plateforme



BTB Branch Target Buffer, structure de prédiction des cibles de branchement

BTI Branch Target Injection, injection de cible de branchement

CFI Control-Flow Integrity, intégrité du flot de contrôle

CLKSCREW Nom de l'attaque publiée sous le titre *Exposing the Perils of Security-Oblivious Energy Management*

CVE Common Vulnerabilities and Exposures, identifiant public de vulnérabilité

DMA Direct Memory Access, accès direct à la mémoire par un périphérique

DRAM Dynamic Random-Access Memory, mémoire vive dynamique

DRAMMER Deterministic Rowhammer Attacks on Mobile Platforms, attaque Rowhammer sur plateformes mobiles

DVFS Dynamic Voltage and Frequency Scaling, ajustement dynamique de la tension et de la fréquence

ECC Error-Correcting Code, code correcteur d'erreurs

FPGA Field-Programmable Gate Array, circuit logique programmable

IBPB Indirect Branch Prediction Barrier, barrière de prédiction des branchements indirects

IBRS Indirect Branch Restricted Speculation, restriction de spéculation des branchements indirects

IOMMU Input-Output Memory Management Unit, unité de gestion mémoire des entrées-sorties

ISA Instruction Set Architecture, architecture du jeu d'instructions

KASLR Kernel Address Space Layout Randomization, randomisation de l'espace d'adressage du noyau

KPTI Kernel Page-Table Isolation, isolation des tables de pages du noyau

L1 Level 1, niveau de cache le plus proche du cœur

L1TF L1 Terminal Fault, famille de fautes transitoires associées au cache L1

LTO Link-Time Optimization, optimisation à l'édition de liens

LVI Load Value Injection, injection de valeur de chargement

MDS Microarchitectural Data Sampling, échantillonnage de données microarchitecturales

MMU Memory Management Unit, unité de gestion mémoire

OS Operating System, système d'exploitation

PCID Process-Context Identifier, identifiant de contexte de processus

RIDL Rogue In-Flight Data Load, attaque par échantillonnage de données en vol

RISC Reduced Instruction Set Computer, ordinateur à jeu d'instructions réduit

RISC-V Architecture ouverte de jeu d'instructions de la famille RISC

RSB Return Stack Buffer, tampon de prédiction des adresses de retour

RTOS Real-Time Operating System, système d'exploitation temps réel

SGX Software Guard Extensions, technologie d'enclaves matérielles d'Intel

SMT Simultaneous Multithreading, exécution simultanée de plusieurs fils sur un cœur

SoC System on Chip, système intégré sur une puce

SSBD Speculative Store Bypass Disable, désactivation du contournement spéculatif des écritures

STIBP Single Thread Indirect Branch Predictors, isolation des prédicteurs indirects entre fils

TEE Trusted Execution Environment, environnement d'exécution de confiance

TLB Translation Lookaside Buffer, cache des traductions d'adresses

TRR Target Row Refresh, rafraîchissement ciblé des rangées DRAM

USENIX Association organisatrice de conférences et éditrice de publications en informatique

VM Virtual Machine, machine virtuelle

2

Introduction et périmètre

2.1 Objectif du guide

Ce document explique les attaques microarchitecturales aux ingénieurs qui conçoivent, intègrent ou sécurisent des systèmes embarqués. Il décrit les mécanismes physiques et logiciels qui rendent ces attaques possibles, présente les principales familles publiées depuis 2014, et replace les contre-mesures dans leur contexte : niveau matériel, système d'exploitation, compilateur et application.

Le sujet ne se limite pas à une vulnérabilité particulière d'un processeur. Il s'agit d'une classe de problèmes où une implémentation optimisée révèle, par un effet observable, des informations que le modèle architectural devait garder inaccessibles. Les exemples les plus connus exploitent la spéculation, les caches ou la DRAM ; d'autres exploitent les prédicteurs, les buffers internes ou la consommation électrique.

Le guide n'est pas une procédure d'exploitation. Les mécanismes sont décrits de manière à permettre l'analyse de risque, la revue d'architecture et la préparation d'une réponse produit, sans fournir de code d'attaque prêt à l'emploi.

2.2 Public visé

Le guide s'adresse :

- aux équipes de développement logiciel embarqué et système
- aux architectes, responsables de lot et responsables produit
- aux équipes de sécurité évaluant des SoC, des RTOS, des hyperviseurs ou des TEE
- aux équipes de validation et de réponse aux vulnérabilités

2.3 Périmètre et terminologie

Les attaques retenues sont des attaques documentées publiquement ayant contribué à faire évoluer la compréhension de la sécurité des processeurs. Elles couvrent les processeurs généralistes et certains systèmes embarqués qui intègrent des cœurs d'application, des caches ou une DRAM

externe. La mention d’une attaque ne signifie pas que tous les processeurs d’une famille sont vulnérables : la vulnérabilité dépend de l’implémentation, de la révision du silicium, du microcode, du logiciel et des conditions d’accès.

La littérature utilise des numérotations parfois divergentes. « Spectre v1 » et « Spectre v2 » désignent les deux classes originales les plus courantes ; les suffixes et numéros ultérieurs ne sont pas une taxonomie universelle. Les dates indiquées ci-dessous sont les premières dates de publication ou de divulgation publique identifiables, et non la date de découverte privée. Pour Rowhammer, le mot consacré dans la littérature est *Rowhammer*. « RawHammer » n’est pas une famille d’attaque microarchitecturale distincte couramment reconnue ; le guide interprète donc cette mention comme Rowhammer.

2.4 Résumé exécutif

Trois idées résument la plupart des attaques présentées :

1. l’annulation architecturale d’une instruction n’efface pas nécessairement les traces laissées dans les structures internes
2. un attaquant peut transformer une différence de latence, de cache ou de données en canal d’observation
3. une correction logicielle peut réduire le risque sans supprimer le comportement matériel qui en est la cause

Pour l’embarqué, le premier geste n’est donc pas d’appliquer indistinctement un correctif associé à un nom d’attaque. Il faut identifier les cœurs réellement présents, leurs mécanismes d’exécution, les frontières de privilège et de confiance, le partage des ressources, le modèle de mise à jour et les mesures d’atténuation supportées par le fournisseur.

3

Fondamentaux microarchitecturaux

3.1 Architecture visible et implémentation réelle

Une ISA définit le comportement fonctionnel attendu d'un processeur. Elle ne spécifie généralement pas les temps d'accès exacts à la mémoire, la politique détaillée du cache ou l'état des prédicteurs. Deux cœurs qui exécutent les mêmes instructions peuvent donc produire le même résultat architectural tout en ayant des comportements temporels différents.

Les performances dépendent notamment de caches de données et d'instructions, de TLB, de prédicteurs de branchement, de préchargements, de files de chargement/stockage et, pour les systèmes avec DRAM, de la gestion des banques et des lignes. Ces structures conservent temporairement un état dépendant des accès précédents. Dès lors qu'un acteur peut le mesurer ou le perturber, il peut devenir un canal entre domaines qui semblaient isolés.

3.1.1 Structures internes utiles à l'analyse

Pipeline et renommage de registres. Le front-end récupère et décode les instructions, puis les place dans des structures qui permettent leur exécution parallèle. Le renommage associe les registres architecturaux à des registres physiques afin de supprimer certaines dépendances artificielles. Les résultats attendent généralement dans un *reorder buffer* avant d'être retirés dans l'ordre du programme. Ce retrait ordonné garantit une vue architecturale cohérente, mais n'efface pas les modifications déjà apportées à des structures comme le cache.

Prédicteurs de branchement. Le processeur mémorise l'historique des branches conditionnelles, les cibles de branches indirectes et parfois les retours. Il peut utiliser plusieurs tables indexées par des bits d'adresse et d'historique. L'algorithme exact est propre à chaque microarchitecture et rarement totalement documenté. Il faut donc distinguer le nom générique d'une structure de son implémentation réelle, notamment pour déterminer si deux domaines partagent ou peuvent influencer son état.

Caches et TLB. Un cache est organisé en lignes, en ensembles et en voies. Plusieurs adresses mémoire correspondent au même ensemble, ce qui crée de la contention même lorsqu'elles désignent des données distinctes. Les caches de données ne sont pas les seules structures pertinentes : les caches d'instructions, les caches de traduction et le TLB peuvent aussi révéler des accès ou créer de la contention. La latence mesurée dépend du niveau de cache, du contrôleur mémoire, de la fréquence et de l'activité concurrente.

Files de chargement et de stockage. Elles suivent les accès mémoire en attente et les écritures qui n'ont pas encore été propagées. Elles servent à résoudre les dépendances et à détecter si une charge peut réutiliser une écriture antérieure. Certaines attaques ciblent leurs valeurs ou leurs états transitoires, plutôt que le contenu permanent d'un cache.

DRAM et contrôleur mémoire. La mémoire principale est organisée en banques, rangées et colonnes. L'ouverture d'une rangée peut accélérer les accès suivants à cette rangée, mais fermer ou ouvrir une autre rangée engendre une latence différente. Le contrôleur ordonnance les requêtes, applique des politiques de rafraîchissement et peut proposer des mécanismes de prévention des perturbations. Ces choix ont un effet à la fois sur les performances et sur la sécurité.

3.2 Exécution hors ordre et spéculation

Un cœur moderne peut décoder plusieurs instructions à la fois, les exécuter hors ordre lorsque leurs opérandes sont prêts, puis retirer leurs résultats dans l'ordre architectural. Une dépendance mémoire lente ou un branchement difficile à prédire augmente l'intérêt de l'exécution spéculative : le cœur travaille en avance au lieu d'attendre.

Si une prédiction est fautive, les résultats architecturaux du chemin transitoire sont abandonnés. En revanche, des effets déjà appliqués à un cache, à un prédicteur ou à une file interne peuvent persister assez longtemps pour être mesurés. Une attaque de type Spectre fait exécuter transitoirement un calcul qui dépend d'une donnée protégée, puis encode cette donnée dans un effet mesurable. L'attaquant n'a pas besoin que le calcul transitoire soit validé architecturalement.

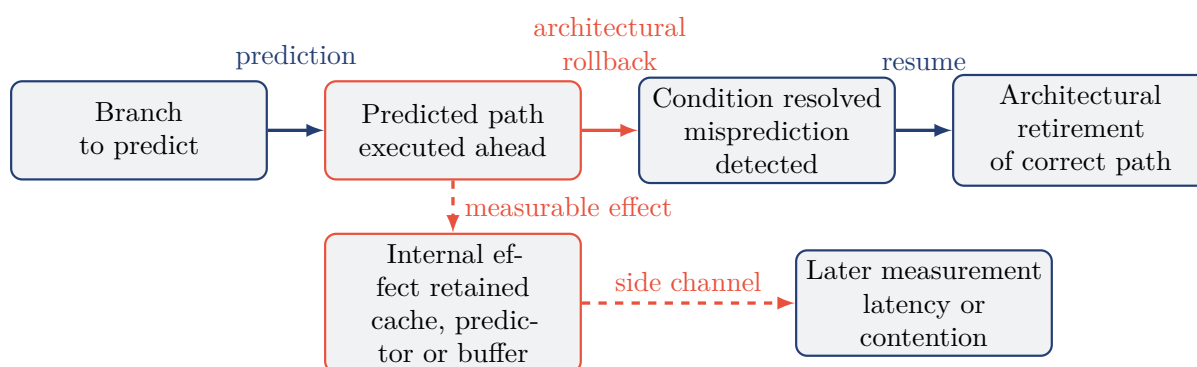


FIGURE 3.1 – Une instruction transitoire peut disparaître de l'état architectural tout en laissant une trace dans l'état microarchitectural

3.2.1 Pourquoi la retraite ordonnée ne suffit pas

Le retrait en ordre protège la cohérence architecturale des exceptions et du flot de contrôle. Par exemple, une charge interdite peut déclencher une exception précise sans que son résultat ne

soit écrit dans un registre architectural visible. Cette garantie ne promet pas que les accès faits pendant l'exécution hors ordre soient invisibles pour toutes les ressources matérielles.

Il faut distinguer trois états. Le premier est l'état architectural qui subsiste après le retrait. Le deuxième est l'état transitoire des instructions en vol, qui est annulé lors d'une mauvaise prédiction. Le troisième est l'état microarchitectural durable ou semi-durable, comme une ligne de cache, qui peut survivre à l'annulation et influencer une mesure ultérieure. Les attaques transitoires exploitent l'écart entre le deuxième et le troisième état.

La durée utile de cet état dépend du chemin de résolution. Une faute de page, un défaut de cache ou une chaîne de dépendances peut retarder la validation d'une instruction. L'attaquant cherche une fenêtre suffisamment longue pour qu'une instruction dépendant d'une donnée protégée atteigne une unité d'exécution puis modifie une structure observable. Plus la latence à attendre est importante, plus le risque d'exécution transitoire augmente, mais les processeurs imposent aussi des limites de ressources et de profondeur de spéculation.

Tous les processeurs ne spéculent pas de la même manière. Un microcontrôleur simple, en ordre et sans prédicteur sophistiqué, n'offre pas nécessairement les primitives de Spectre décrites pour un cœur d'application. Il peut néanmoins rester concerné par d'autres canaux (temps, consommation, contention de bus, erreurs de mémoire ou DMA).

3.3 Canaux auxiliaires usuels

Temps d'accès au cache. Une ligne déjà présente dans le cache est généralement servie plus vite qu'une ligne absente. Flush+Reload et Prime+Probe ont démontré des méthodes de mesure exploitant respectivement des lignes partagées et l'occupation de jeux de cache [36, 21].

Contention et latence. Un attaquant peut mesurer la compétition sur un cache, un bus mémoire, un port d'exécution, une banque DRAM ou un prédicteur, sans partager directement l'adresse observée.

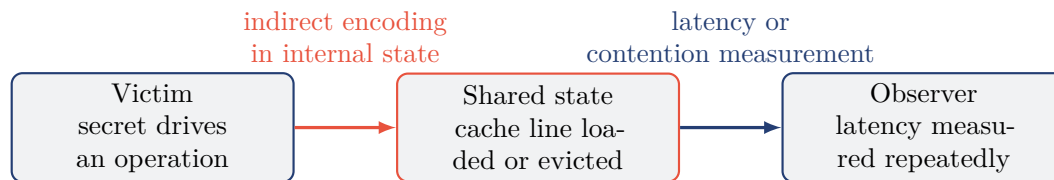
Consommation et fréquence. Le courant, la puissance et les variations de fréquence peuvent dépendre de l'activité des unités. Hertzbleed a montré en 2022 que la mise à l'échelle dynamique de fréquence pouvait transformer des variations de puissance dépendantes des données en variations temporelles observables à distance dans certaines conditions [34].

Perturbation plutôt que mesure. Rowhammer se distingue des canaux précédents : l'effet recherché est une modification de bits dans la mémoire, et non seulement l'inférence d'une donnée par son temps d'accès.

3.3.1 Deux familles de mesure du cache

Flush+Reload. La méthode est applicable lorsqu'un même contenu physique est partagé, par exemple une bibliothèque partagée. L'observateur retire une ligne du cache, laisse la victime s'exécuter, puis mesure le temps nécessaire pour relire cette ligne. Un accès rapide suggère que la victime l'a remise en cache. Les pages dédoublées, les bibliothèques partagées et les enclaves utilisant des pages communes peuvent rendre le partage possible. La méthode observe une ligne précise avec une bonne résolution, mais requiert ce partage et un moyen de mesurer les latences.

Prime+Probe. Cette méthode n'exige pas que les deux domaines partagent une ligne mémoire. L'observateur accède à un ensemble d'adresses qui occupent les voies d'un ensemble de cache, laisse la victime s'exécuter, puis mesure le coût de ses propres accès répétés. Si la victime a utilisé le même ensemble, certaines lignes de l'observateur peuvent avoir été évincées. L'attaque dépend de la connaissance ou de l'inférence de la fonction d'indexation, de la politique de remplacement et de la stabilité des latences.



The visible result is not the secret itself
but a distribution of measurements that depends on the cache state

FIGURE 3.2 – Principe d'un canal auxiliaire fondé sur un état partagé du cache

Bruit et qualité de mesure. Les interruptions, les changements de fréquence, les autres cœurs, le préchargement matériel et le système d'exploitation ajoutent du bruit. Un attaquant peut répéter les observations et les agréger, ce qui explique pourquoi réduire la précision d'un minuteur ne suffit pas toujours. Le risque dépend aussi de la répétabilité du secret, de la fréquence à laquelle l'opération peut être déclenchée et du nombre d'observations autorisé.

Canaux sans partage de lignes. Des canaux de contention peuvent exploiter des ressources partagées, même si les adresses sont disjointes. L'interférence sur des bancs mémoire, des unités d'exécution, des TLB ou des files internes peut fournir une mesure plus grossière. Une analyse de menace doit donc distinguer « pas de mémoire partagée » de « pas de ressource partagée ».

3.4 Chaîne générale d'une attaque transitoire

Un scénario conceptuel comporte quatre étapes :

1. **Conditionner** Le contexte ou le code place une structure interne comme un prédicteur dans un état choisi
2. **Déclencher** Une mauvaise prédiction, un défaut ou une dépendance lente ouvre une fenêtre d'exécution transitoire
3. **Encoder** Le chemin transitoire utilise une donnée protégée pour influencer un état microarchitectural distinctif
4. **Observer** Une mesure du temps ou de la contention permet de distinguer les états et de reconstituer l'information

Cette représentation est utile pour raisonner, mais les attaques réelles diffèrent dans leurs préconditions, leur primitive d'encodage et leur portée. Une fuite ne constitue pas automatiquement une exécution de code arbitraire ; elle peut toutefois révéler des secrets cryptographiques, des données de noyau, des informations d'un autre processus ou des données d'une machine virtuelle.

3.4.1 Ce qui distingue une fuite d'une compromission complète

Une mesure répétée peut d'abord révéler un bit ou quelques bits. La reconstruction d'une clé complète demande souvent que l'opération cryptographique soit reproductible, que l'attaquant puisse provoquer ou attendre un nombre suffisant d'exécutions et que le bruit reste maîtrisable. Une fuite mémoire peut exposer des données sans donner le contrôle du flot d'exécution. Une corruption Rowhammer peut modifier une page, mais l'attaquant doit encore transformer cette modification en effet exploitable.

Les rapports de recherche démontrent une chaîne sous des hypothèses précises. Pour transposer le résultat à un produit, il faut vérifier l'existence du gadget ou du chemin cible, les privilèges disponibles à l'attaquant, les ressources réellement partagées, la précision temporelle et la possibilité de répéter l'expérience. Un CVE lié à une famille de processeurs ne décrit donc pas à lui seul la gravité de chaque configuration.

4

Panorama des attaques publiées

4.1 Repères chronologiques

- 2006 – 2014 : canaux de cache** Les travaux de Osvik, Shamir et Tromer sur le cache et de Yarom et Falkner sur Flush+Reload établissent des techniques de mesure réutilisées par de nombreuses attaques [21, 36]
- 2014 – 2016 : Rowhammer** L'étude de Kim et al. formalise les inversions de bits DRAM dues à des activations répétées en 2014. Les travaux suivants démontrent des scénarios JavaScript, mobiles et systèmes [10, 27, 32]
- Janvier 2018 : Spectre et Meltdown** Les divulgations de Spectre et Meltdown rendent publiques les conséquences de l'exécution spéculative sur l'isolation entre processus et entre niveaux de privilège [11, 17]
- 2018 : variantes précoces** Spectre v1.1 et v1.2 explorent les écritures spéculatives et la protection de la mémoire en lecture seule. Foreshadow démontre des fuites depuis des enclaves SGX et d'autres contextes L1 [5, 31]
- 2019 : accès mémoire et buffers** MDS, RIDL, Fallout et ZombieLoad montrent qu'il est possible d'échantillonner des données de buffers microarchitecturaux, y compris à travers certaines frontières de sécurité [6, 33, 24, 26]
- 2017 : ClockScrew** Cette attaque montre qu'un contrôle logiciel de la gestion énergétique peut provoquer des fautes de calcul dans un domaine sécurisé TrustZone [28]
- 2020 – 2021 : nouvelles primitives** LVI inverse le sens de la fuite en injectant une valeur dans un calcul transitoire. Les recherches Rowhammer font évoluer les motifs d'activation et les moyens de contourner les défenses [30, 3, 13, 9]
- 2022 – 2024 : évolutions récentes** Retbleed, BHI, Inception et Downfall explorent respectivement les retours, l'historique de branchement et les instructions de collecte vectorielle. Zenbleed touche une implémentation AMD précise [35, 19, 16, 20, 1]

Les dates de divulgation ne se confondent pas toujours avec celles des actes de conférence. Les équipes de recherche coordonnent souvent la publication avec les fournisseurs, puis publient l'article scientifique plusieurs mois plus tard. Les dates de la chronologie sont celles mentionnées dans les avis ou articles correspondants et non des dates de disponibilité générale des correctifs.

4.2 Spectre : exécution spéculative contrôlée

4.2.1 Spectre v1 – contournement spéculatif de vérification

La première variante publiée de Spectre, en janvier 2018, exploite typiquement une vérification de borne ou un autre contrôle conditionnel. Un attaquant influence le prédicteur pour qu'il anticipe le chemin « autorisé », alors que la condition réelle ne sera connue qu'ensuite. Pendant la fenêtre transitoire, le code utilise un index qui dépend d'une donnée hors du domaine permis. Un second accès indexé laisse une empreinte dans le cache ; après résolution de la condition, le programme architecturalement correct a pourtant déjà modifié l'état interne [11].

L'impact n'est pas intrinsèquement lié au noyau : un site web peut attaquer un processus navigateur dans certaines configurations ; un processus peut viser des données accessibles au même espace d'adressage ou à des composants co-résidents. L'exploitation dépend du code cible, de la précision des mesures et des frontières effectivement partageant les ressources.

Conditions de réussite. Il faut un chemin de code victime qui lise ou transforme une donnée sensible avant qu'une vérification soit définitivement résolue. L'attaquant doit pouvoir provoquer une prédiction utile, influencer au moins une entrée du chemin spéculatif et mesurer une ressource qui dépend du résultat. Le gadget peut être dans une bibliothèque, un interpréteur, un noyau ou un firmware. Il n'est pas nécessaire que l'attaquant injecte des instructions dans le code victime, mais il doit pouvoir influencer les données ou le contexte qui activent le gadget.

Exemple conceptuel. Un service vérifie qu'un index appartient à un tableau public. Si cette vérification est mal prédite, une lecture transitoire peut utiliser l'index non validé et obtenir une valeur située hors du tableau. Une opération ultérieure transforme cette valeur en choix entre plusieurs lignes de mémoire. Une fois la branche corrigée, l'observateur compare les latences et en déduit une information sur la valeur. Le point crucial est que la vérification protège le résultat architectural, mais n'empêche pas nécessairement l'exécution spéculative du corps.

Portée et variantes de gadget. La variante v1 est souvent décrite comme un contournement de vérification de borne, mais cette formulation est plus étroite que la classe réelle. Des conditions de privilège, des valeurs nulles, des tailles, des pointeurs ou d'autres contrôles de flot peuvent être mal prédits. La valeur divulguée et le canal d'encodage ne sont pas nécessairement dans le même objet mémoire. Cette diversité explique pourquoi une recherche de séquences de code connues ne couvre pas tous les gadgets possibles.

Les premières réponses ont inclus des barrières de spéculation placées après les vérifications sensibles, des séquences de masquage d'index dont la sûreté est démontrée pour l'ISA, et des techniques de compilation telles que Speculative Load Hardening [18, 15]. Une vérification classique dans le langage C n'est pas automatiquement une barrière microarchitecturale : le compilateur et le processeur peuvent continuer à spéculer. Les barrières doivent être choisies selon le processeur et insérées aux points où l'information protégée influence un accès observable. Les variantes Spectre v1.1 et v1.2, publiées en 2018, ont montré que l'analyse doit aussi considérer les écritures spéculatives, les gadgets de stockage et les pages marquées en lecture seule [5].

Limites des réponses logicielles. Une barrière posée trop tôt peut être déplacée ou éliminée si le compilateur ne la reconnaît pas comme primitive de synchronisation. Une barrière posée trop tard peut laisser passer un accès dépendant du secret. Le masquage d'index dépend du type, de la largeur et de la sémantique exacte de l'ISA. Il doit être revu après changement de compilateur, d'options d'optimisation ou de microarchitecture. Un test fonctionnel qui ne vérifie que la valeur de retour ne détecte pas une fuite transitoire.

4.2.2 Spectre v2 – injection de cible de branchement

Spectre v2, également appelé Branch Target Injection (BTI), manipule les prédictions de branchement indirect. Une victime exécute un branchement indirect ; le prédicteur, influencé par un contexte antérieur, propose une cible contrôlée ou utile à l'attaquant. Si le chemin prédit exécute transitoirement un « gadget » qui dépend d'un secret, ce secret peut être divulgué via un canal auxiliaire. Le partage de certains prédicteurs entre contextes rend possibles des attaques entre processus ou niveaux de privilège, selon l'implémentation [11].

Prédiction de cible et gadget. Un branchement indirect est un saut dont la cible est calculée à l'exécution, par exemple un appel de fonction à travers un pointeur. Le processeur tente de prédire sa destination à partir des branches précédentes. Le BTB peut associer l'adresse du branchement à une cible prédite. Un attaquant entraîne l'état de prédiction dans un contexte, puis provoque l'exécution du branchement victime. Si le prédicteur réutilise un état partagé ou insuffisamment séparé, le cœur peut commencer à exécuter le gadget avant de vérifier la véritable cible.

Isolation entre contextes. Le processus d'exploitation dépend de la façon dont la microarchitecture indexe les tables de prédiction, du moment où elles sont nettoyées et des transitions de privilège. Les domaines n'ont pas besoin de partager le même espace d'adressage si la prédiction transitoire peut viser du code victime accessible dans le domaine de confiance. En virtualisation, l'ordonnanceur et le partage du cœur physique déterminent si deux invités peuvent influencer des structures communes.

Les défenses initiales ont combiné retpoline (transformation du branchement indirect par le compilateur), IBRS/IBPB/STIBP sur certaines générations x86, et des barrières ou séquences de protection fournies par les fabricants [29, 15]. Ces mécanismes ont un coût variable en performance et en consommation. Ils ne sont pas interchangeables : retpoline ne couvre pas toutes les formes de prédiction et son efficacité dépend de la microarchitecture.

Pourquoi plusieurs mitigations coexistent. Retpoline modifie le code généré pour éviter certaines prédictions de branchement indirect. IBRS contrôle le comportement de prédiction en fonction du niveau de privilège sur les processeurs qui le supportent. IBPB demande une barrière de prédiction lors d'une transition choisie. STIBP limite le partage de certains prédicteurs entre fils SMT. La documentation de chaque processeur précise les combinaisons sûres, les microcodes nécessaires et le coût. Une option activée dans le noyau n'implique pas que chaque chemin de transition utilise toutes ces primitives.

Des attaques ultérieures ont démontré que le domaine d'historique ou l'état du prédicteur pouvait rester pertinent malgré les protections conçues autour des cibles directes. Branch History Injection (BHI), divulguée en 2022 et publiée en 2024, montre qu'un historique de branchement peut influencer une prédiction indirecte dans des contextes où des défenses antérieures semblaient suffisantes. Retbleed (2022) montre que des instructions de retour peuvent, sur certaines

microarchitectures x86, se comporter comme des branchements indirects vulnérables. Les correctifs correspondants ont requis de nouveaux choix de mitigation, des microcodes et parfois des modifications des noyaux [35, 19].

Évolution des contournements. Les mécanismes récents tentent de séparer ou d'effacer non seulement une cible prédite, mais aussi l'historique de branches qui contribue à choisir le gadget. BHI souligne qu'un contrôle des cibles ne suffit pas forcément si l'historique global reste influençable. Retbleed a conduit à revoir la manière dont certains processeurs prédisent les retours et à ajouter des séquences logicielles ou microcodes spécifiques. Inception montre qu'un modèle de menace doit inclure les chemins d'entraînement atypiques, et pas seulement les branches explicitement présentes dans le code source. Ces résultats illustrent une évolution de la recherche, et non une preuve que toutes les mitigations sont inefficaces.

4.2.3 Autres variantes Spectre

Spectre v4 / Speculative Store Bypass (2018). Le processeur peut avoir une écriture en attente dans sa file de stockage sans encore connaître son adresse ou sa donnée complète. Pour éviter de bloquer, il peut exécuter une charge plus récente en supposant que cette charge n'accède pas à la même adresse. Si l'hypothèse est fausse, la charge a lu une ancienne valeur. Un calcul qui dépend de cette valeur peut ensuite laisser une trace dans le cache avant que le processeur ne détecte la dépendance et rejoue les instructions nécessaires. Les processeurs concernés proposent notamment des mécanismes de désactivation du contournement spéculatif (SSBD). Leur activation ciblée ou globale peut pénaliser les performances [8].

Spectre v1.1 et v1.2 (2018). La variante v1.1 étudie des écritures spéculatives déclenchées par un contournement de vérification de borne. Une écriture transitoire peut modifier temporairement une cible ou une structure utilisée ensuite par la victime. La variante v1.2 s'intéresse à des protections de lecture seule qui ne bloquent pas nécessairement une écriture spéculative déjà en vol sur les processeurs étudiés. Ces propositions ont élargi la revue des gadgets au-delà des charges mémoire et ont souligné que les permissions architecturales ne décrivent pas toujours le comportement avant résolution [5].

Spectre v3a / Rogue System Register Read (2018). Cette variante lit transitoirement des registres système privilégiés sur certaines implémentations, puis encode la valeur dans un état microarchitectural. Elle diffère de la lecture de mémoire noyau classiquement associée à Meltdown, même si les deux peuvent s'appuyer sur un défaut de privilège transitoire. La mitigation documentée pour une variante ne peut être généralisée sans vérifier l'avis du fabricant et les instructions accessibles à l'attaquant [7].

SpectreRSB (2018). L'attaque vise le Return Stack Buffer (RSB), qui prédit les cibles des instructions de retour. Un appel alimente cette structure avec l'adresse de retour attendue et l'instruction de retour la consomme. Si la pile de retours est vide, débordée ou contaminée lors d'une transition de contexte, le processeur peut se rabattre sur d'autres prédicteurs ou utiliser un état résiduel. Cette mauvaise prédiction peut amener la victime à exécuter transitoirement un gadget. Les défenses incluent le remplissage ou le nettoyage du RSB lors de transitions pertinentes, mais le comportement et les séquences de mitigation sont propres aux familles de processeurs [12].

NetSpectre (2019). Cette étude montre qu'un canal de type Spectre peut, dans des conditions spécifiques, être exploité à distance au travers d'un service réseau. La victime doit exposer un chemin de code adéquat et l'attaquant doit pouvoir répéter ses requêtes assez souvent pour distinguer un signal du bruit réseau. Les débits rapportés sont faibles par rapport à une mesure locale, et la stabilité du service est déterminante. Le résultat montre que l'accès à un minuteur local précis n'est pas toujours indispensable, mais ne signifie pas que tout service réseau est exploitable [25].

Inception (2023). Cette attaque contre des processeurs AMD spécifiques combine des comportements de prédiction et une technique dite de « phantom speculation » pour créer des chemins transitoires inattendus, y compris autour des retours. Elle illustre qu'une mitigation établie pour une famille de gadgets ne couvre pas nécessairement toutes les voies d'entraînement du prédicteur [16].

Définition des numéros. Les noms Spectre v1 à v4 proviennent de publications différentes et ne couvrent pas toutes les attaques de prédiction. Les appellations v1.1, v1.2 et v3a décrivent des raffinements ou des variantes proposés par des chercheurs et des fournisseurs. Retbleed, BHI et Inception sont généralement présentés sous leur nom propre plutôt que comme un numéro Spectre suivant. Pour déterminer une mitigation, il faut donc partir du bulletin constructeur et de la microarchitecture, puis relier ce bulletin à la famille d'attaque décrite dans la publication.

4.3 Meltdown et fuites liées aux fautes

4.3.1 Meltdown (2018)

Meltdown a été divulguée publiquement en janvier 2018 et décrite en détail par Lipp et al. La vulnérabilité exploitée sur certaines générations de processeurs permet à une charge d'accéder transitoirement à une donnée dont la lecture sera ensuite déclarée interdite. La faute est précise du point de vue architectural, mais survient trop tard pour empêcher certains effets internes produits par la charge. Une seconde opération dépend de la donnée lue et modifie l'état du cache. L'attaquant mesure ensuite les latences et associe les lignes présentes à des valeurs possibles [17].

Scénario et impact. Dans le scénario historique étudié, un processus non privilégié répétait les lectures sur des adresses du noyau mappées dans son espace virtuel mais protégées par les permissions. Les mesures ont montré qu'une information pouvait être reconstruite octet par octet dans les conditions de l'expérience. La disponibilité de KASLR n'annule pas le problème, car il peut d'abord être possible de découvrir des données servant à localiser ou comprendre l'espace noyau. Le risque dépend toutefois du modèle exact de processeur et de la présence du mapping concerné.

Différence avec Spectre v1. Meltdown n'exige pas nécessairement un gadget de vérification mal prédit dans le logiciel victime. La faute de privilège elle-même ouvre la fenêtre transitoire exploitée. Spectre v1, à l'inverse, utilise une mauvaise prédiction du flot de contrôle pour contourner transitoirement un contrôle dans un code qui s'exécute normalement. Les deux familles utilisent un canal auxiliaire, mais elles diffèrent par la condition de déclenchement et la localisation du gadget.

La réponse logicielle majeure sur les systèmes concernés a été KPTI (Kernel Page-Table Isolation), qui sépare davantage les mappings du noyau et de l'espace utilisateur. Des microcodes et de nouvelles conceptions de processeurs ont également contribué à corriger la classe de comportement [14, 15]. KPTI ajoute des transitions et peut avoir un coût, variable selon la présence de PCID et la charge.

Limites de KPTI. KPTI retire la majorité des mappings noyau de la table utilisée en mode utilisateur, ce qui réduit la surface accessible par l'attaque originale. Il ne transforme pas tous les accès mémoire en opérations non spéculatives, et ne constitue donc pas une réponse générale à Spectre. La bascule entre tables de pages peut accroître le coût des appels système et des interruptions. L'effet est atténué sur certaines plateformes par des mécanismes de gestion des identifiants de contexte, mais doit être mesuré sur le système embarqué réel.

Autres variantes Meltdown. Les travaux ultérieurs ont étudié des lectures transitoires de données liées à des registres système, à des protections de superviseur et à des accès en mode utilisateur [17, 31]. Les noms attribués à ces variantes ne désignent pas une vulnérabilité uniforme présente sur tous les processeurs. Ils décrivent des résultats obtenus sur des modèles et des conditions précis. Les versions du firmware et du microcode, les permissions de page et les changements de mode doivent être documentés pour chaque produit.

4.3.2 Foreshadow / L1TF et failles voisines

Foreshadow, publié en 2018, montre qu'une lecture transitoire peut révéler des données présentes dans le cache L1 dans des circonstances impliquant des défauts de traduction ou de privilège. Les variantes démontrées concernent notamment des enclaves Intel SGX et des environnements virtualisés ; la portée dépend des contrôles d'isolation, du partage de cœur et du logiciel de virtualisation [31].

Les mitigations ont compris la mise à jour du microcode, le vidage ou la gestion du cache L1 lors de transitions pertinentes, ainsi que des recommandations aux hyperviseurs sur l'ordonnancement et l'allocation de mémoire [31, 6]. L'approche rejoint Meltdown par l'exploitation d'un état transitoire, mais les hypothèses et la ressource ciblée diffèrent ; le correctif de l'un ne prouve pas l'absence de l'autre.

4.4 Échantillonnage de données internes : MDS et dérivés

En 2019, plusieurs travaux publiés sous le nom de Microarchitectural Data Sampling (MDS) ont étudié des données transitoires présentes dans des buffers internes aux processeurs Intel : buffers de chargement, de stockage ou de lignes de remplissage. RIDL, Fallout et ZombieLoad en décrivent différentes voies d'échantillonnage et scénarios. Contrairement au modèle élémentaire de Spectre, l'attaquant ne choisit pas toujours directement l'adresse de la donnée divulguée ; il échantillonne des valeurs qui transitent par une structure interne. Des informations de processus, de noyau ou de machine virtuelle ont pu être récupérées dans les conditions de laboratoire rapportées [6, 33, 24, 26].

Les mesures ont inclus des mises à jour de microcode qui ajoutent des instructions de vidage de buffers, leur invocation aux transitions de sécurité et, dans certains environnements, la

désactivation du SMT [6]. Le coût dépend des transitions et du workload. Ces attaques sont fortement dépendantes du modèle de cœur ; leur nom ne doit pas être extrapolé à des cœurs ARM ou RISC-V sans avis spécifique du fabricant.

Différences entre sous-familles. Les dénominations MDS regroupent des recherches sur l'échantillonnage de données de structures internes. RIDL décrit des lectures de données en vol provenant de différentes structures et chemins de chargement. Fallout s'intéresse notamment aux buffers de stockage. ZombieLoad vise des données qui transitent par les buffers de remplissage et montre des scénarios de fuite entre contextes. Les frontières franchissables et les conditions requises varient selon les processeurs et les configurations SMT.

Pourquoi le contenu n'est pas toujours choisi. Dans une attaque MDS, l'échantillonnage peut retourner une valeur laissée par un accès récent ou un mélange de valeurs transitoires. L'attaquant doit souvent répéter l'opération et filtrer les résultats. Cela diffère d'un gadget Spectre soigneusement construit où une donnée choisie par le secret peut influencer directement un index de cache. Les fuites sont alors plus bruitées, mais peuvent toucher des informations d'autres contextes logiques.

Réponse opérationnelle. Les microcodes ont introduit des opérations permettant de vider certaines structures, mais le logiciel doit les utiliser aux frontières prévues. Le noyau choisit parfois une stratégie de nettoyage à chaque changement de contexte, uniquement lors de transitions sensibles ou lors du passage vers une machine virtuelle. La désactivation du SMT peut réduire le partage concurrent, au prix d'une baisse de débit et d'une consommation par tâche différente. Le profil de mesure choisi doit correspondre à la politique réellement activée en production.

Downfall (2023). Downfall a révélé une fuite transitoire sur certaines générations de processeurs Intel via l'instruction vectorielle Gather et des buffers de registres temporaires. Les résultats ont motivé un correctif microcode et une analyse d'impact des charges utilisant intensivement ces instructions ; la mitigation peut avoir un coût mesurable dans certains usages vectoriels [20].

Zenbleed (2023). Zenbleed est une vulnérabilité spécifique à certaines générations de processeurs AMD Zen 2. Un comportement de renommage de registres vectoriels peut permettre à un processus d'observer des données d'un autre contexte logique. Ce cas montre qu'une fuite de données microarchitecturale ne se réduit pas nécessairement à un canal cache classique. Le correctif recommandé est un microcode fourni par AMD ; l'impact précis dépend du processeur et de la version installée [1].

4.5 Injection de valeur et exécution spéculative par gadgets

Load Value Injection (LVI), divulguée en 2020 puis publiée en 2021, inverse le modèle de plusieurs attaques précédentes. Au lieu de faire sortir une valeur secrète par un canal auxiliaire, l'attaquant cherche à injecter une valeur contrôlée ou influencée dans une opération transitoire de la victime, puis à détourner son calcul spéculatif. Les démonstrations ont visé notamment des environnements SGX et peuvent nécessiter une chaîne d'exploitation exigeante. Les mesures sont

spécifiques aux plateformes, et les barrières ou séquences de défense peuvent induire un coût élevé dans certaines enclaves [30].

Chaîne conceptuelle. L’attaquant provoque une situation où une charge transitoire reçoit une valeur provenant d’une structure interne influencée par un autre contexte. La valeur injectée modifie le calcul spéculatif d’une victime de confiance. Si ce calcul accède ensuite à une ressource en fonction de l’opérande, le résultat peut être divulgué par un canal secondaire. Cette attaque est donc une injection de donnée dans un calcul, non un simple échantillonnage passif.

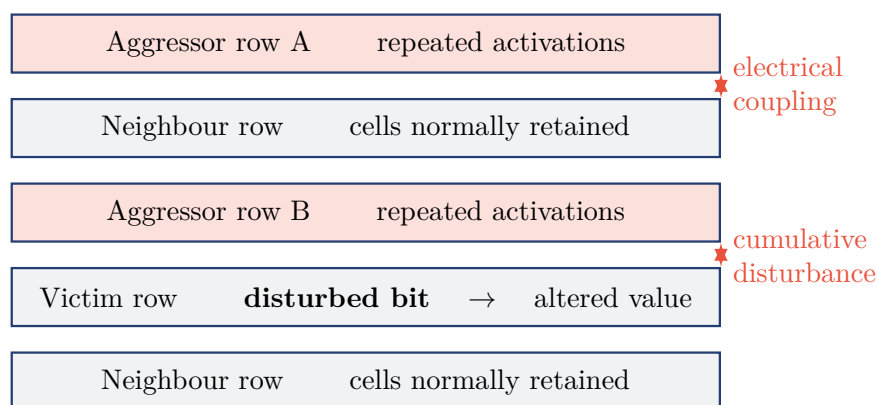
Conséquences et mitigation. LVI peut transformer une enclave supposée protéger les secrets en cible d’un gadget, sous les conditions décrites par la publication. Les atténuations peuvent imposer des barrières autour des charges sensibles et augmenter considérablement le temps d’exécution dans les environnements enclavés. L’évaluation doit déterminer si la charge utile de confiance contient des gadgets concernés et si le fournisseur recommande la mitigation pour le modèle de processeur utilisé. Un mécanisme conçu contre les fuites MDS ne doit pas être présumé suffisant contre une injection ciblée.

Ces résultats soulignent qu’une stratégie uniquement fondée sur « nettoyer les caches après usage » est incomplète : l’attaquant peut agir sur des valeurs d’opérandes ou des structures internes en amont du calcul.

4.6 Rowhammer et perturbations DRAM

4.6.1 Principe et premières démonstrations

En 2014, Kim et al. ont documenté des inversions de bits dans la DRAM lorsque des lignes voisines sont activées de façon répétée. Les marges électriques des cellules étant limitées, une forte activité de lignes dites « agresseurs » peut perturber une ligne voisine dite « victime ». Les premières démonstrations ont été suivies par des attaques de bout en bout : Rowhammer.js a montré une voie depuis un navigateur en 2016, tandis que DRAMMER a étudié l’exploitation sur des appareils mobiles Android de la même période [10, 27, 32].



The controller may refresh DRAM at regular intervals without necessarily counting adversarial activations of each row

FIGURE 4.1 – Une activité répétée sur des lignes agresseurs peut perturber des cellules d’une ligne victime

De l'accès virtuel à l'activation physique. Un programme demande des lectures ou écritures via des adresses virtuelles. Le système d'exploitation traduit ces adresses en pages physiques, puis le contrôleur mémoire répartit les requêtes dans les rangées et banques DRAM. L'attaquant doit donc obtenir un motif d'accès qui atteint fréquemment les rangées physiques visées. Les caches peuvent empêcher certaines requêtes de parvenir à la DRAM, et les préchargeurs peuvent modifier le motif. Les travaux historiques ont étudié différents moyens de réduire l'effet du cache ou de créer une pression mémoire suffisante.

Conditions pratiques. La possibilité de provoquer une inversion dépend du type de DRAM, de la géométrie interne, de la température, du rafraîchissement, du contrôleur et des paramètres de chronométrage. La cartographie entre adresse physique et rangée n'est pas toujours publique. Même si des inversions sont possibles, l'attaquant doit placer des données sensibles dans la rangée victime ou tirer parti d'une structure logicielle dont la corruption est utile. La fiabilité des bits inversés et leur polarité sont des contraintes importantes dans la construction d'une attaque.

Une inversion de bit n'est pas à elle seule une compromission. Il faut obtenir un emplacement utile, prédire ou détecter le bit affecté, puis faire en sorte que la corruption modifie un objet exploitable. Des travaux tels que Flip Feng Shui et RAMBleed ont montré des impacts sur des systèmes et des données sensibles ; RAMBleed est particulièrement important car il transforme des perturbations DRAM en canal de lecture sous certaines hypothèses [23, 13].

Impact sur l'isolation mémoire. Les démonstrations ont visé des tables de pages, des métadonnées de gestion mémoire, des clés ou des fichiers mis en cache. La corruption d'une entrée de table de pages peut changer la traduction ou les permissions si la disposition mémoire et le bit inversé s'y prêtent. Flip Feng Shui étudie le placement de données et de pages pour rendre certains changements exploitables. DRAMMER a démontré que des contraintes de gestion mémoire mobile n'excluaient pas toute exploitation. Ces résultats dépendent de l'OS, de l'allocateur et du contrôleur.

RAMBleed comme canal de lecture. RAMBleed n'est pas uniquement une attaque de corruption. Les auteurs exploitent une relation statistique entre les valeurs de cellules adjacentes et le comportement de perturbation pour inférer des bits d'une ligne voisine. La fuite nécessite des conditions particulières de placement, de répétition et de filtrage. Elle montre toutefois que la disponibilité d'une mémoire ECC ne transforme pas automatiquement toute perturbation en simple erreur corrigée et invisible [13].

4.6.2 Contourner ou dépasser les défenses

Les premières défenses reposaient notamment sur le rafraîchissement, des seuils de comptage d'activations (TRR), le placement mémoire et la mémoire ECC. Des travaux ultérieurs ont montré leurs limites :

- TRRespass (2020) et Blacksmith (divulgué en 2021 puis publié en 2022) ont produit des motifs complexes et non uniformes qui contournent certaines implémentations de Target Row Refresh
- Half-Double (2021) a montré que l'effet peut s'étendre au-delà des seules lignes immédiatement adjacentes, selon la DRAM et les conditions
- ECCploit (2019) a étudié la possibilité de dépasser ou contourner la protection ECC dans certaines configurations

- RowPress (2023) a étudié l'effet du maintien prolongé d'une ligne activée plutôt que la seule répétition rapide d'activations

Ces travaux ne prouvent pas que toutes les mémoires ou tous les contrôleurs sont vulnérables. Ils démontrent en revanche qu'un mécanisme de mitigation interne, opaque et dépendant du fabricant ne constitue pas une garantie universelle [3, 9, 4, 2, 22].

4.6.3 Mesures adaptées aux produits embarqués

Les mesures possibles incluent le choix de DRAM et de contrôleurs qualifiés, le maintien à jour des paramètres et révisions recommandés, une protection ECC correctement dimensionnée, l'activation des fonctions de suivi/rafraîchissement disponibles, la limitation des accès mémoire non fiables et la réduction des privilèges des composants qui peuvent générer une forte activité. L'ECC corrige certaines erreurs simples, mais n'est pas une parade absolue face à des motifs adverses, des erreurs multiples ou des attaques par canal auxiliaire. Les essais doivent correspondre au composant DRAM et au contrôleur réellement déployés.

À vérifier lors de la qualification. Le dossier matériel doit identifier le fabricant, la référence, la révision de DRAM, le contrôleur mémoire et les paramètres de rafraîchissement. Les fonctions de suivi d'activation doivent être configurées selon le profil de mémoire, et leur fonctionnement doit être confirmé sur le matériel livré. La politique de remplacement et les changements de source d'approvisionnement doivent déclencher une réévaluation, car deux composants compatibles sur le plan électrique peuvent présenter des comportements de perturbation différents.

4.7 CLKSCREW : fautes provoquées par la gestion énergétique

4.7.1 Pourquoi cette attaque entre dans le guide

CLKSCREW, publiée par Adrian Tang, Simha Sethumadhavan et Salvatore Stolfo au symposium USENIX Security 2017, n'est pas une attaque de canal auxiliaire et ne repose pas sur une fuite transitoire comparable à Spectre. Elle appartient d'abord à la famille des attaques par injection de fautes. Elle mérite néanmoins sa place dans une analyse microarchitecturale au sens large, car elle détourne les mécanismes internes de gestion de fréquence et de tension d'un processeur pour modifier le résultat de calculs exécutés par un domaine de confiance [28].

La distinction est importante pour la terminologie. Ici, la microarchitecture n'est pas seulement mesurée par l'attaquant, comme dans un canal de cache. Son point de fonctionnement électrique et temporel est manipulé jusqu'à ce que certaines opérations ne respectent plus leurs marges de fonctionnement. La propriété attaquée est l'intégrité du calcul, plutôt que la confidentialité par observation.

4.7.2 Rappel sur DVFS et marges temporelles

La gestion dynamique de la tension et de la fréquence (DVFS) ajuste le point de fonctionnement du processeur à la charge afin d'économiser de l'énergie ou de maîtriser la température. Une fréquence plus élevée réduit le temps disponible pour stabiliser les signaux dans chaque cycle. Une tension plus faible ralentit certaines portes logiques. Le matériel et le micrologiciel doivent donc maintenir des couples tension-fréquence validés, souvent nommés points de fonctionnement.

La marge temporelle est la différence entre le délai maximal du chemin logique le plus lent et la période d'horloge disponible. Si la période devient trop courte pour le délai réel, un registre peut capturer une valeur instable ou incorrecte. Le processeur n'a pas nécessairement une détection ou une correction générale de ce défaut. La faute peut alors se propager dans un calcul cryptographique, une vérification ou une décision de contrôle.

4.7.3 Principe de CLKSCREW

Les systèmes évalués par les auteurs exposaient à du logiciel du monde normal des interfaces de gestion énergétique suffisamment puissantes pour influencer le fonctionnement du processeur. L'attaquant disposait d'un pilote noyau malveillant dans le monde normal Android. Il manipulait le point de fonctionnement pendant que du code sensible s'exécutait dans ARM TrustZone. En poussant le processeur hors des conditions de fonctionnement sûres, il provoquait des erreurs de calcul sans accès physique ni équipement externe d'injection de fautes.

L'intérêt de l'attaque vient du fait que TrustZone isole normalement les logiciels sécurisés au moyen des niveaux d'exécution et des contrôles matériels. Pourtant, une ressource électrique ou un mécanisme DVFS commun au système peut être piloté depuis un domaine moins privilégié si l'interface logicielle n'applique pas les frontières de sécurité. Le domaine sécurisé conserve ses permissions mémoire, mais il ne peut pas garantir la justesse du matériel si un autre domaine modifie son point de fonctionnement.

4.7.4 Impacts démontrés

Les auteurs ont démontré sur des appareils ARM/Android commerciaux qu'un pilote noyau malveillant pouvait extraire des clés cryptographiques protégées par TrustZone. Ils ont également montré une élévation de privilèges permettant de charger du code non signé dans le domaine TrustZone. Ces résultats ne signifient pas que tout processeur ARM ou toute implémentation TrustZone soit vulnérable. Ils dépendent des contrôles d'accès aux interfaces énergétiques, de l'organisation des régulateurs et des mécanismes de détection d'erreurs du SoC testé [28].

Le prérequis est plus fort qu'un simple compte applicatif : dans le scénario publié, l'attaquant contrôle un pilote noyau du monde normal. Cette position reste réaliste dans un modèle où le noyau riche peut être compromis par une vulnérabilité logicielle ou par une mise à jour malveillante. CLKSCREW transforme alors une compromission du monde normal en atteinte à l'intégrité et à la confidentialité du domaine sécurisé.

4.7.5 Mesures de protection et limites

La première mesure consiste à traiter la fréquence, la tension et les régulateurs comme des ressources de sécurité. Les interfaces de programmation DVFS ne doivent pas être accessibles sans médiation à un pilote non fiable. Le moniteur sécurisé, le micrologiciel ou un contrôleur matériel doit imposer les points de fonctionnement validés, les limites de fréquence et les changements permis pendant l'exécution d'un service de confiance.

Les mesures complémentaires comprennent la vérification des transitions DVFS en présence de tâches sécurisées, l'isolation ou l'arbitrage des régulateurs partagés, la détection de violations de tension et de fréquence, ainsi que des contrôles d'intégrité sur les calculs sensibles. Une routine cryptographique peut parfois détecter une faute par redondance ou vérification de résultat, mais ces mécanismes doivent être conçus contre un attaquant capable de provoquer des erreurs adaptatives. Une seconde exécution identique ne suffit pas si le fauteur peut cibler les deux exécutions de manière corrélée.

Un verrouillage logiciel est utile uniquement s'il est appliqué par une racine de confiance et ne peut pas être modifié depuis un domaine de privilège inférieur. Il faut aussi vérifier les chemins de déverrouillage, de reprise après veille, de mise à jour de firmware et de changement de mode de performance. L'avis constructeur et la documentation du SoC restent déterminants, car les limites tension-fréquence et les contrôles de régulateur sont propres à chaque composant.

4.8 Canaux temporels hors exécution transitoire

Toutes les attaques microarchitecturales ne reposent pas sur une mauvaise prédiction. Les canaux de cache exploités contre des implémentations cryptographiques sont antérieurs à Spectre et Meltdown. Un chiffrement peut par exemple consulter des tables dont l'adresse dépend d'un octet de clé. Si les temps d'accès diffèrent selon que la ligne est présente ou non dans le cache, un observateur peut accumuler des mesures et éliminer les clés incompatibles avec les observations. L'étude de Osvik, Shamir et Tromer a notamment analysé les attaques de cache et leurs contre-mesures pour AES [21].

Hertzbleed (2022). Les processeurs ajustent leur fréquence en fonction de la puissance et de leurs limites thermiques. Hertzbleed montre qu'un effet de puissance dépendant des données peut, dans certaines conditions, modifier la fréquence de fonctionnement et donc le temps d'exécution observé à distance. Une protection classique contre les mesures locales de consommation n'est pas suffisante si un service distant permet de répéter une opération sensible et de mesurer sa durée [34].

Mesures défensives. Les implémentations à temps constant évitent que les chemins de contrôle ou les adresses mémoire dépendent directement des secrets. Il faut aussi tenir compte de la fréquence dynamique, des instructions utilisées, du code généré et des possibilités de mesures répétées. Les algorithmes masqués ou blinding rendent parfois les observations moins directement exploitables, sans constituer une preuve de temps constant. La validation doit être menée sur le processeur final, avec les fréquences et les conditions thermiques du produit. Les contremesures et leurs limites sont décrites dans les travaux sur les attaques de cache et Hertzbleed [21, 34].

5

Impacts sur un système embarqué

5.1 Lecture transversale des familles d'attaque

Ce tableau sert d'orientation, pas de verdict de conformité. Une mitigation peut couvrir plusieurs attaques, mais son effet dépend toujours des révisions matérielles, des versions logicielles et des transitions concernées. À l'inverse, la présence d'une faiblesse dans un processeur n'implique pas nécessairement une voie exploitable dans le produit si aucun secret ou gadget pertinent n'est accessible à l'attaquant.

5.2 Frontières de sécurité concernées

Selon la plateforme, un attaquant peut tenter de franchir une ou plusieurs frontières :

- processus utilisateur vers processus co-résident
- utilisateur vers noyau, hyperviseur ou firmware
- invité vers hyperviseur ou autre machine virtuelle
- environnement riche vers TEE ou enclave
- application distante vers service local ou distant exposant un canal

Le partage d'un cœur physique, du SMT, d'un cache de dernier niveau, d'un prédicteur ou d'une DRAM influence le risque. Un TEE protège l'accès architectural à la mémoire ; il ne garantit pas, à lui seul, l'absence de canaux auxiliaires dans le cœur, le cache ou les interfaces mémoire.

5.3 Conséquences et facteurs de risque

Les conséquences vont de la récupération d'un secret ponctuel à la fuite répétée de données inter-processus ou inter-VM, en passant par l'élévation de privilèges lorsque la fuite est combinée

Famille	État exploité	Effet recherché	Défense prioritaire
Spectre v1	Prédiction de branche conditionnelle	Exécution transitoire d'un gadget activé par des données contrôlées	Barrières ou masquage validés autour des accès sensibles
Spectre v2, BHI	Prédiction de cible et historique de branchement	Détournement transitoire d'un branchement indirect ou d'un retour	Microcode, séquences du compilateur et isolation de prédicteurs selon le processeur
Meltdown, Fore-shadow	Chargement transitoire malgré une faute ou un défaut de traduction	Lecture d'une donnée privilégiée encore accessible au chemin microarchitectural	Correction silicium et séparation des mappings avec KPTI si requis
MDS, Downfall, Zenbleed	Buffers ou registres temporaires	Échantillonnage ou exposition de données d'un autre contexte	Microcode et nettoyage des structures documentés par le fournisseur
LVI	Valeur de charge injectée dans un calcul transitoire	Influence du calcul d'un contexte de confiance	Séquences spécifiques aux plateformes et revue des gadgets
Rowhammer	État physique des cellules DRAM	Inversion ou inférence de bits en mémoire principale	DRAM qualifiée, rafraîchissement et protection ECC adaptés
CLKSCREW	Point de fonctionnement tension-fréquence du processeur	Fautes de calcul dans un domaine sécurisé	Accès DVFS médié et points de fonctionnement protégés
Canaux temporels	Cache, puissance, fréquence ou contention	Inférence statistique d'une donnée sensible	Algorithmes à temps constant et limitation du partage exploitable

TABLE 5.1 – Les familles diffèrent par la primitive matérielle et demandent des mitigations adaptées

avec une autre faiblesse. Les facteurs déterminants incluent l'exposition à du code non fiable, l'accès à un minuteur ou à un grand nombre de mesures, le partage matériel, la présence de secrets en mémoire, l'activation d'un hyperviseur ou d'un TEE, et la durée pendant laquelle le produit ne peut pas être mis à jour.

Confidentialité. C'est l'impact le plus direct des canaux de cache et des attaques transitoires. Les données concernées peuvent être des clés, des jetons d'authentification, des contenus utilisateur ou des secrets de firmware. La sensibilité dépend de leur durée de présence en mémoire et de la possibilité de les renouveler. Une clé éphémère qui reste quelques millisecondes peut rester exposée si un attaquant peut déclencher le chemin répété.

Intégrité. Rowhammer peut modifier une donnée ou un contrôle de sécurité si le bit inversé touche une structure critique. Une valeur corrompue peut contourner une permission, changer un pointeur ou altérer du code chargé dynamiquement. Cette conséquence exige un placement utile et une corruption répétable. Les attaques d'injection transitoire comme LVI influencent un calcul spéculatif sans nécessairement changer l'état architectural validé.

Disponibilité et sûreté de fonctionnement. Une mitigation peut allonger le temps de réponse, réduire le débit, provoquer plus de transitions entre tables ou imposer la désactivation du SMT. Dans un produit temps réel, la question n'est pas seulement « le correctif est-il actif ? », mais aussi « les bornes de latence restent-elles garanties ? ». Une réponse de sécurité qui casse une échéance critique peut créer un risque de sûreté. Les tests doivent donc mesurer les pires cas représentatifs et vérifier les budgets de temps et d'énergie.

Un processeur « embarqué » n'est pas automatiquement immunisé. Les cœurs d'application ARM utilisés dans les passerelles, téléphones, cartes de calcul et systèmes industriels sont susceptibles d'intégrer spéculation et caches complexes. À l'inverse, il ne faut pas attribuer à un Cortex-M, un cœur RISC-V ou un SoC donné une vulnérabilité connue sur x86 sans confirmer la microarchitecture et l'avis constructeur. Une simple étiquette ISA ne suffit pas.

5.4 Particularités de l'embarqué

5.4.1 Cœurs asymétriques et SoC hétérogènes

Un même SoC peut associer des cœurs de performance et des cœurs économes en énergie, voire des générations de cœur différentes. Le noyau peut migrer une tâche d'un type de cœur à un autre selon la charge ou la température. Les prédicteurs et caches peuvent être privés à un cœur, partagés dans un cluster ou partagés à l'échelle du SoC. Une mesure effectuée sur un seul cœur ne permet donc pas nécessairement de caractériser tous les chemins d'exécution.

L'inventaire doit descendre au niveau des cœurs physiques et des révisions. Il est également utile de relever le partage du cache de dernier niveau, la présence de SMT, les politiques de migration et les mécanismes d'isolation proposés par le fournisseur. Les processeurs configurables ou les FPGA avec cœurs intégrés peuvent avoir des versions de cache et de prédicteur différentes suivant l'image de synthèse.

5.4.2 Temps réel et systèmes à ressources contraintes

Les systèmes temps réel peuvent limiter les accès concurrents, verrouiller des lignes de cache ou éviter la migration des tâches. Ces choix réduisent parfois le bruit des mesures, mais peuvent aussi rendre les temps d'exécution plus prévisibles pour un observateur. Les interruptions périodiques et les tâches de fond peuvent servir d'horloge ou de point de synchronisation même lorsqu'un minuteur de haute résolution n'est pas accessible.

Les mitigations ont un coût mesurable dans les chemins critiques. Une barrière peut augmenter la latence d'un traitement d'interruption. Un vidage de cache peut retarder le retour à une tâche. La désactivation du SMT peut réduire le parallélisme et modifier la consommation. La validation doit donc combiner l'analyse de fuite, la mesure de latence pire cas et la vérification des échéances fonctionnelles.

5.4.3 DMA, accélérateurs et périphériques

Un moteur DMA ou un accélérateur peut accéder à la mémoire pendant que le processeur exécute une tâche. Les caches non cohérents, les buffers de translation d'un IOMMU et la contention du bus créent des états partagés supplémentaires. Ces ressources ne sont pas nécessairement visées par Spectre, mais peuvent révéler ou perturber des accès. Une périphérie capable de générer un débit important peut également affecter les mesures temporelles ou les mécanismes de rafraîchissement DRAM.

Le modèle de menace doit identifier les maîtres de bus, les régions mémoire accessibles à chacun, les règles de cohérence et les contrôles de l'IOMMU. L'isolation de processus par MMU n'isole pas automatiquement les accès DMA. Les pilotes, firmwares de périphériques et moteurs de chiffrement doivent être examinés comme des composants pouvant influencer la mémoire et les temps d'accès.

5.4.4 Démarrage, mise à jour et durée de vie

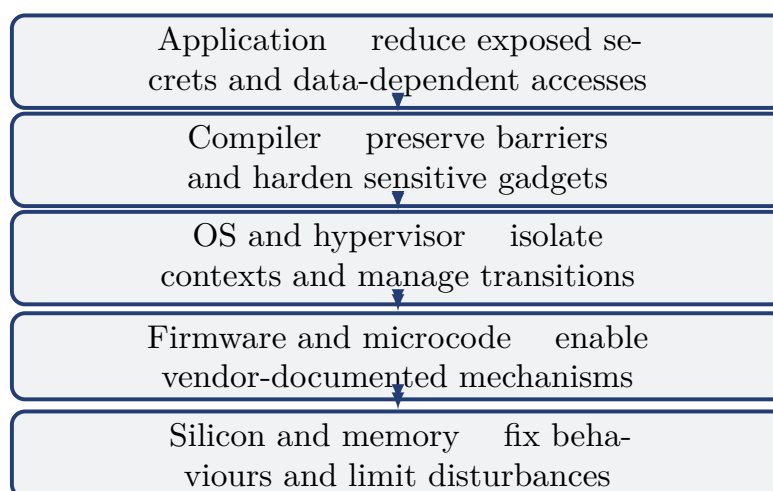
Un système embarqué peut rester déployé plusieurs années sans maintenance fréquente. Le mécanisme de démarrage sécurisé doit alors authentifier les mises à jour du noyau, du firmware et du microcode, et empêcher le retour à une version qui réactive une faiblesse connue. La chaîne de mise à jour doit pouvoir déployer une mitigation tout en préservant un mode de récupération signé.

Les équipements isolés du réseau ne sont pas nécessairement hors de portée. Un attaquant local, un code tiers, une interface de maintenance, une mise à jour compromise ou une machine virtuelle invitée peut constituer le point d'entrée. L'évaluation doit distinguer le mode de livraison du correctif, la possibilité d'une mise à jour pendant l'exploitation et la capacité d'un équipement à vérifier que la nouvelle mitigation est effectivement active.

6

Contre-mesures et méthode d'analyse

6.1 Défense en profondeur



Security depends on the alignment of all layers
and on the configuration actually shipped

FIGURE 6.1 – La mitigation doit rester cohérente du code applicatif jusqu’au matériel

Une stratégie efficace combine des contrôles qui agissent à des niveaux différents. L’application réduit les gadgets et les secrets exposés. Le compilateur maintient les séquences de protection. Le noyau et l’hyperviseur traitent les transitions de contexte. Le firmware active le microcode et configure les fonctions du SoC. Le silicium et la DRAM doivent enfin fournir les propriétés attendues. Une faille de configuration à une seule couche peut annuler le bénéfice des autres.

6.1.1 Silicium et microcode

Les corrections matérielles peuvent modifier le comportement fautif, isoler des prédicteurs, effacer des buffers ou ajouter un contrôle de spéculation. Elles offrent la réponse la plus proche de

la cause, mais ne couvrent pas nécessairement tous les gadgets ou effets analogues. Le microcode peut déployer des mécanismes sur du matériel existant, à condition que le fournisseur et la plateforme le permettent. Vérifier la version courante est nécessaire : installer une image système ne garantit pas que le microcode requis ait été chargé [15, 6].

Vérification en production. Le microcode peut être chargé par le BIOS, par un chargeur de démarrage ou dynamiquement par l'OS. La chaîne de démarrage, les politiques de mise à jour et les permissions de l'image influencent donc la version réellement exécutée. Le produit doit conserver une preuve de la version active et éviter un retour à une version vulnérable lors d'une restauration ou d'une mise à jour de firmware.

6.1.2 Noyau, hyperviseur et firmware

Les systèmes peuvent appliquer KPTI, des barrières, des mécanismes de nettoyage de prédicteur ou de buffers, des politiques d'ordonnancement et des options de désactivation du SMT. Un firmware sécurisé doit considérer les transitions vers et depuis les domaines moins fiables, les chemins d'exception, les changements de contexte et les modes de veille/réveil. Les politiques d'atténuation peuvent dépendre du processeur et être désactivées ou restreintes par configuration : il faut documenter cet état et pas seulement la présence d'un correctif dans le code source.

Barrière mémoire ou barrière de spéculation. Ces deux termes ne sont pas synonymes. Une barrière mémoire ordonne les lectures et écritures visibles pour les cœurs ou les périphériques. Elle peut ne pas empêcher une instruction dépendante d'être exécutée transitoirement. Une barrière de spéculation ou un séquençement recommandé par le fabricant vise la progression spéculative et a une sémantique propre à l'ISA. Sur ARM, x86 et RISC-V, les noms d'instructions, garanties et limitations diffèrent. Il faut suivre la documentation de la famille de cœur et ne pas remplacer une barrière par une instruction qui ne garantit que l'ordre mémoire.

Transitions souvent oubliées. La revue doit inclure les changements de tâche, les changements de niveau de privilège, les interruptions, les exceptions synchrones, les retours de veille, les changements d'invité et les entrées ou sorties de TEE. Un chemin de transition atypique peut omettre le nettoyage appliqué au chemin normal. Les séquences de mise à jour de microcode doivent aussi être vérifiées après redémarrage à froid et lors d'un redémarrage logiciel.

6.1.3 Compilateur et application

Les barrières et séquences de masquage doivent être comprises par le compilateur et garanties par l'ISA. Une construction de langage de haut niveau ne devient pas une barrière matérielle par intention. Les mises à jour du compilateur peuvent introduire ou supprimer des séquences de protection selon les options. Les routines qui manipulent des secrets doivent privilégier des algorithmes à temps constant et éviter les accès dépendant des données lorsque cela est pertinent ; ces mesures n'empêchent pas à elles seules toutes les attaques transitoires.

Validation du binaire livré. Les audits de source ne prouvent pas à eux seuls que le binaire contient les barrières voulues. Il faut inspecter l'assembleur produit pour les fonctions sensibles, vérifier les options LTO et d'optimisation, et maintenir des tests de non-régression lors des mises à jour du compilateur. Les attributs, intrinsèques ou macros de barrière ont des sémantiques spécifiques à l'architecture. Une instruction de barrière mémoire destinée à ordonner les accès n'est pas nécessairement une barrière contre la spéculation.

Gestion des secrets. La réduction de la durée de vie des clés, la séparation des secrets entre tâches et la suppression des copies inutiles peuvent diminuer l'exposition. Les algorithmes cryptographiques à temps constant évitent généralement les branchements ou accès mémoire indexés par des données secrètes, mais ils ne garantissent pas l'absence de fuite par fréquence, consommation ou contention. Il faut examiner le code généré sur la cible et prendre en compte les accélérateurs cryptographiques ainsi que leurs buffers et DMA.

6.1.4 Mémoire et DRAM

Pour Rowhammer, la défense doit inclure DRAM, contrôleur, micrologiciel et charges logicielles. Les mécanismes TRR, ECC et rafraîchissement peuvent varier selon le fabricant, la densité mémoire, les températures et la révision. La spécification d'une fonctionnalité ne garantit pas que sa configuration de production soit active ni que ses propriétés résistent à des motifs adaptatifs.

ECC et intégrité logicielle. L'ECC peut corriger certaines erreurs simples et signaler des erreurs plus complexes, mais la couverture dépend du code utilisé, de la largeur des mots et de l'organisation des barrettes. Les contrôleurs peuvent masquer, corriger ou journaliser différemment les erreurs. Un produit critique doit définir une réaction au compteur d'erreurs, aux erreurs corrigées répétées et aux erreurs non corrigibles. Une correction silencieuse n'est pas un substitut à la détection d'une activité anormale.

6.2 Évolutions et limites des mitigations

Les contre-mesures sont parfois contournées parce qu'elles répondent à une instance plutôt qu'à la primitive générale. Une protection qui bloque une cible BTB n'efface pas nécessairement l'historique global; une barrière après une vérification ne couvre pas un autre gadget situé plus tôt; un mécanisme de rafraîchissement calibré sur un motif uniforme peut manquer des activations irrégulières; le nettoyage d'un buffer lors d'une transition donnée ne protège pas une autre transition.

Les divulgations de Retbleed, BHI, Inception et Blacksmith sont des exemples de cette évolution. Elles ne rendent pas toutes les mitigations précédentes inutiles, mais imposent de vérifier précisément leur hypothèse de sécurité, les variantes de matériel qu'elles couvrent et les mises à jour qui les complètent. L'état de mitigation doit être suivi par référence de processeur et de firmware, pas par une simple liste de noms d'attaques.

6.2.1 Comment analyser un contournement annoncé

Une nouvelle publication ne signifie pas toujours que tous les systèmes précédemment corrigés redeviennent vulnérables. Il faut distinguer le mécanisme scientifique, le matériel effectivement testé et les conditions nécessaires à l'exploitation. Une étude peut démontrer un contournement sur une microarchitecture particulière, sans montrer qu'il s'applique à chaque produit utilisant la même ISA.

1. **Identifier la primitive** Déterminer si la publication cible une cible de branchement, un historique, un buffer, un cache, un chemin de traduction ou la DRAM
2. **Lire le modèle d'attaquant** Vérifier les droits initiaux, les possibilités de co-résidence, le nombre de mesures et les interfaces accessibles

3. **Comparer les révisions matérielles** Relever les cœurs et révisions testés ainsi que les microcodes et options de firmware utilisés
4. **Relier le résultat au correctif** Vérifier si le fournisseur a publié un avis, une mise à jour, une configuration recommandée ou une déclaration de non-applicabilité
5. **Évaluer le scénario produit** Déterminer la présence de code non fiable, de secrets exposés et d'un canal mesurable sur le produit réel
6. **Tracer la décision** Consigner les hypothèses, les versions testées, les mesures activées et les risques résiduels

Cette méthode évite deux erreurs opposées. La première consiste à déclarer tous les appareils vulnérables parce qu'un processeur de la même famille est concerné. La seconde consiste à ignorer un avis parce que le nom de l'attaque n'apparaît pas dans une ancienne matrice de conformité.

6.3 Procédure de qualification pour un produit

1. **Inventorier** Relever les références exactes des SoC, cœurs, révisions, microcodes, DRAM, contrôleurs, compilateurs, noyaux, hyperviseurs et TEE
2. **Définir les frontières** Cartographier les secrets et les domaines qui les partagent : processus, cœurs, caches, prédicteurs, buffers, bus et DRAM
3. **Consulter les avis primaires** Utiliser les bulletins du fabricant, les avis des mainteneurs du système et les références de CVE correspondant à la référence matérielle réelle
4. **Établir l'état actif** Vérifier les versions de microcode et de logiciel, les indicateurs de mitigation rapportés par le système, les paramètres de compilation, l'ordonnancement et la configuration SMT
5. **Évaluer la charge et les coûts** Mesurer les pénalités de performance, latence, consommation et démarrage induites par les mesures retenues sur les scénarios représentatifs du produit
6. **Tester et surveiller** Utiliser les tests du fournisseur et une stratégie de validation reproductible, puis intégrer les nouveaux avis dans la gestion de vulnérabilités et le processus de mise à jour

6.4 Questions de revue de conception

- Des données secrètes sont-elles présentes dans un domaine co-résident avec du code non fiable ?
- Les barrières requises sont-elles spécifiques à l'architecture et conservées après optimisation ?
- Les transitions de privilège, d'exception, de veille ou de virtualisation nettoient-elles l'état nécessaire ?
- Le TEE ou l'hyperviseur partage-t-il des caches, prédicteurs ou buffers avec des domaines adverses ?
- Les mitigations sont-elles activées dans l'image et dans les conditions de production ?

- Le produit peut-il recevoir les microcodes, firmwares et correctifs système pendant toute sa durée de vie ?
- Les protections DRAM sont-elles vérifiées sur les références de mémoire effectivement achetées ?
- Les coûts de mitigation sont-ils compris et acceptés par l'architecture du produit ?

7

Conclusion

Les attaques microarchitecturales ont établi que la correction architecturale ne suffit pas à démontrer la confidentialité. Les caches, prédicteurs, buffers, unités vectorielles et mémoires DRAM peuvent introduire des effets observables ou perturbables en dehors du modèle fonctionnel habituel.

Pour les ingénieurs embarqués, une réponse robuste combine l'identification exacte du matériel, une analyse des frontières de confiance, l'application des mitigations recommandées par les fournisseurs, des pratiques logicielles adaptées et une stratégie de mise à jour durable. Les noms Spectre, Meltdown ou Rowhammer sont des repères historiques utiles, non des tests binaires de sécurité. La propriété à établir est plus précise : sur cette révision matérielle, dans cette configuration logicielle et face à ce modèle d'attaquant, les secrets et les opérations sensibles sont-ils protégés par des mécanismes effectivement activés et vérifiés ?

Bibliographie

- [1] Advanced Micro Devices. Amd-sb-7008 : Transient scheduler attack. Security Bulletin, 2023. URL : <https://www.amd.com/en/resources/product-security/bulletin/amd-sb-7008.html>.
- [2] Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. Exploiting correcting codes : On the effectiveness of ECC memory against rowhammer attacks. In *2019 IEEE Symposium on Security and Privacy*, 2019. doi:10.1109/SP.2019.00089.
- [3] Pietro Frigo, Enrique Meyer, Ke Xiao, Harish Chintala, Victor van der Veen, Mahyar Ghasempour, Moinuddin K. Qureshi, Khaled N. Khasawneh, and Daniel Gruss. TRRespass : Exploiting the many sides of target row refresh. In *2020 IEEE Symposium on Security and Privacy*, 2020. URL : <https://www.vusec.net/projects/trrespass/>.
- [4] Google. Half-double : Next-generation rowhammer. Google Research, 2021. URL : <https://research.google/pubs/half-double-next-generation-rowhammer/>.
- [5] Jann Horn. Speculative execution, variant 1.1 : Bounds check bypass store. Project Zero security research, 2018. URL : <https://googleprojectzero.blogspot.com/2018/05/reading-privileged-memory-with-side.html>.
- [6] Intel. Microarchitectural data sampling. Security advisory, 2019. URL : <https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/data-sampling.html>.
- [7] Intel Product Security Incident Response Team. Rogue system register read. Security advisory for CVE-2018-3640, 2018. URL : <https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00115.html>.
- [8] Intel Product Security Incident Response Team. Speculative store bypass. Security advisory, 2018. URL : <https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00115.html>.
- [9] Patrick Jattke, Michael Gruber, Filip Gąsienica, Fabian Manz, and Kaveh Razavi. Blacksmith : Scalable rowhammering in the frequency domain. In *2022 IEEE Symposium on Security and Privacy*, 2022. URL : <https://www.vusec.net/projects/blacksmith/>.
- [10] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them : An experimental study of DRAM disturbance errors. In *Proceedings of the 41st Annual International Symposium on Computer Architecture*, pages 361–372, 2014. doi:10.1145/2676726.2676970.
- [11] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks : Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy*, pages 1–19, 2019. doi:10.1109/SP.2019.00002.

- [12] Ehsan G. Koruyeh, Khaled Khasawneh, Chengyu Song, and Nael Abu-Ghazaleh. Spectre Returns! speculation attacks using the return stack buffer. arXiv :1807.07940, 2018. URL : <https://arxiv.org/abs/1807.07940>.
- [13] Andrew Kwong, Daniel Genkin, Daniel Gruss, and Yuval Yarom. RAMBleed : Reading bits in memory without accessing them. In *2020 IEEE Symposium on Security and Privacy*, 2020. URL : <https://rambleed.com/>.
- [14] Linux Kernel Documentation. Page table isolation. Linux kernel documentation, 2024. URL : <https://www.kernel.org/doc/html/latest/arch/x86/pti.html>.
- [15] Linux Kernel Documentation. Spectre side channels. Linux kernel documentation, 2024. URL : <https://www.kernel.org/doc/html/latest/admin-guide/hw-vuln/spectre.html>.
- [16] Moritz Lipp et al. Inception : Exposing new attack surfaces with training in transient-execution. In *32nd USENIX Security Symposium*, 2023.
- [17] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown : Reading kernel memory from user space. In *Proceedings of the 27th USENIX Security Symposium*, 2018. URL : <https://www.usenix.org/conference/usenixsecurity18/presentation/lipp>.
- [18] LLVM Project. Speculative load hardening. LLVM Compiler Infrastructure documentation, 2024. URL : <https://llvm.org/docs/SpeculativeLoadHardening.html>.
- [19] Giorgi Maisuradze et al. Branch history injection : On the effectiveness of hardware mitigations against cross-privilege Spectre-v2 attacks. In *33rd USENIX Security Symposium*, 2024. URL : <https://www.usenix.org/conference/usenixsecurity24>.
- [20] Daniel Moghimi. Downfall : Exploiting speculative data gathering. Security research disclosure; subsequently presented at USENIX Security 2024, 2023. URL : <https://downfall.page/>.
- [21] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures : The case of AES. In *Topics in Cryptology – CT-RSA 2006*, 2006. doi:10.1007/11605805_1.
- [22] Jaehyun Park, Jisung Lee, Joonwon Choi, Jaewoo Kim, et al. RowPress : Amplifying read disturbance in modern DRAM chips. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023. doi:10.1145/3579371.3589063.
- [23] Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Flip Feng Shui : Hammering a needle in the software stack. In *Proceedings of the 25th USENIX Security Symposium*, 2016. URL : <https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/razavi>.
- [24] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. Fallout : Leaking data on Meltdown-resistant CPUs. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019. doi:10.1145/3319535.3363219.
- [25] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. NetSpectre : Read arbitrary memory over network. In *Network and Distributed System Security Symposium*, 2019. URL : <https://www.ndss-symposium.org/ndss-paper/netspectre-read-arbitrary-memory-over-network/>.
- [26] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. Zombieload : Cross-privilege-boundary data sampling. In



Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, 2019. doi:10.1145/3319535.3363213.

- [27] Mark Seaborn and Thomas Dullien. Exploiting the DRAM rowhammer bug to gain kernel privileges. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, pages 750–757, 2016. doi:10.1145/2976749.2978406.
- [28] Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo. CLKSCREW : Exposing the perils of security-oblivious energy management. In *Proceedings of the 26th USENIX Security Symposium*, pages 1057–1074, 2017. URL : <https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/tang>.
- [29] Paul Turner. Retpoline : A software construct for preventing branch-target-injection. Google, 2018. URL : <https://support.google.com/faqs/answer/7625886>.
- [30] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Yuval Yarom, Raoul Strackx, Frank Piessens, and Elie Bursztein. Lvi : Hijacking transient execution through microarchitectural load value injection. In *2021 IEEE Symposium on Security and Privacy*, 2021. URL : <https://lviattack.eu/>.
- [31] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Yossi Yee, Elie Bursztein, Jean-Philippe Picod, Kyriakos Kallergi, Kari Kostiaainen, Daniel Gruss, et al. Foreshadow : Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *Proceedings of the 27th USENIX Security Symposium*, 2018. URL : <https://www.usenix.org/conference/usenixsecurity18/presentation/bulck>.
- [32] Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, and Cristiano Giuffrida. Drammer : Deterministic rowhammer attacks on mobile platforms. In *2016 IEEE Symposium on Security and Privacy*, 2016. doi:10.1109/SP.2016.5.
- [33] Stephan Van Schaik et al. Ridl : Rogue in-flight data load. In *2019 IEEE Symposium on Security and Privacy*, 2019. doi:10.1109/SP.2019.00087.
- [34] Yingchen Wang, Martin Kösters, Seán O’Connell, et al. Hertzbleed : Turning power side-channel attacks into remote timing attacks on x86. In *Proceedings of the 31st USENIX Security Symposium*, 2022. URL : <https://www.usenix.org/conference/usenixsecurity22/presentation/wang-yingchen>.
- [35] Ingo Wikner and Kaveh Razavi. Retbleed : Arbitrary speculative code execution with return instructions. In *31st USENIX Security Symposium*, 2022. URL : <https://www.usenix.org/conference/usenixsecurity22/presentation/wikner>.
- [36] Yuval Yarom and Katrina Falkner. FLUSH+RELOAD : A high resolution, low noise, L3 cache side-channel attack. In *Proceedings of the 23rd USENIX Security Symposium*, 2014. URL : <https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom>.

Licence

Ce document est diffuse sous licence **Creative Commons CC BY-NC-ND 4.0**
(Attribution – Pas d’Utilisation Commerciale – Pas de Modification).
Texte integral de la licence : <https://creativecommons.org/licenses/by-nc-nd/4.0/deed.fr>

