

III Attaques micro-architecturales

* **faiblesse implémentation d'une instruction**: l'exemple de **SYSRET**

normalement restaure RIP, le sélecteur de segment de code est modifié: passe de Ring 0 à 3
 ↳ sur **Intel** on vérifie l'@ dans RCX en amont, le CPL est modifié **avant** le chargement de RIP
 ↳ sur **AMD** on ne vérifie pas l'@ dans RCX, le CPL est modifié **après** le chargement de RIP
 ↳ @ dans RCX invalide ⇒ #GP en ring 0 (Intel) ⇒ RSP pas modifié par l'OS mais RSP (et les autres registres) sont déjà restaurés (merci INTEL) ⇒ donc RSP en ring 0 contrôlable par l'utilisateur...
 ↳ **Préparation**: allocation avec mmap d'une page mémoire avant l'@ non canonique limite, à la fin de la zone: ajout d'un bout de code: affecte RAX avec numéro d'appel système souhaité, déclenche SYSCALL (l'instruction est donc à la limite de la zone). préalablement: une @ particulière dans la zone de l'IDT est copiée dans RSP. **Exploitation**: on appelle LE code ⇒ SYSCALL ⇒ fin du traitement ⇒ SYSRET ⇒ #GP ⇒ #PF (car rien de restauré donc GS pointe sur GS user) ⇒ #PF traité par un gestionnaire placé par l'attaquant dans l'espace user.

* **attaque mémoire (RAM)**: Rowhammer (martèlement mémoire)
 ↳ condensateurs non idéaux ⇒ fuite de charges électriques sur les condensateurs voisins
 ↳ **EXPLOIT**: bypasser les caches CPU (flusher) → difficile de tomber sur les bonnes adresses
 ↳ permet de déjauger les JMP, sandbox escape, privileged escalation (bit flip dans PTE)
 ↳ contre-mesures: ECC Memory (mais ECC plat...), Target Row Refresh (mais blacksmith)

* **attaque CPU avec exécution spéculative (out of order)**: Meltdown
 ↳ exécution sans savoir s'il a les droits → exception → **roppback imparfait** → trace
 ↳ ici on attaque la lenteur de vérification des permissions (exécution interdite avant de se raviser)
 ↳ le but est d'attaquer le noyau (vol de secrets)
 ↳ contre-mesure: KPTI (Kernel Page Table Isolation sous Linux) ⇒ on supprime l'espace noyau dans l'espace d'adressage virtuel des processus
 ↳ coût mémoire (2 ensemble de pages), coût exécution (flush TLB, synchro user/kernel)

* **attaque CPU avec exécution spéculative (branchement)**: Spectre
 ↳ bas d'une erreur de flux ou de contrôle → **roppback imparfait** → trace dans le cache
 ↳ ici on exploite la prédiction: on trompe le processeur pour qu'il fasse de mauvaises prédictions
 ↳ le but est d'attaquer un autre processus (vol de secrets)
 ↳ contre-mesure: utilisation de lfence, ou désactivation de l'exécution spéculative dans le registre MSR (config interne du CPU)
 avec microcode intel update (nouveau bit IBRS: Indirect Branch Restricted Speculation), tromperie du prédicteur avec retpoline (prédicteur en boucle infini tant qu'on a pas l'adresse - remplace les CALL/JMP vulnérables).

Injection SMI: injection de code via le mode SHM (super-privilégié)

Branch History Injection: Contourne les protections de Spectre via historique de branchement

ECCploit: Attaque Rowhammer malgré la mémoire à correction d'erreurs

RIDL: Vol des données en transit dans les buffers (ex: LFB), side channel

Fallout: Lecture de données récentes depuis buffer d'écriture (side channel)

FPGAhammer: Attaque Rowhammer depuis une carte FPGA par injection de fautes

PCIleech: Accès mémoire direct et total via périphérique PCIe

Heartbleed: Vol de données via variation de fréquence du CPU (side channel)

Plundervolt: Consomme les calculs en réduisant la tension électrique du CPU
 ↳ par injection de fautes sur Intel SGX
 ↳ attaquent la Dynamic Voltage and Frequency Scaling (DVFS)

IV Protection de base: Ring, trap/call/interrupt, pagination → **insuffisant**

bit **Write Protect (WP)** → bit 16 du CRO

↳ empêche Ring 0 d'écrire dans les sections "read-only" de la Table des pages (TIB)

↳ utile pour copy-on-write (COW): quand on fork(), les pages sont marquées RO si le noyau veut écrire dans du RO, bit WP déclenche une faute, l'OS duplique la page

Execute Disable → au niveau du registre MSR (matériel)

↳ interdit l'exécution de certaines entrées de PDE/PTE (pas d'exécution sur la page)

↳ uniquement en PAE ou Long Mode (64 bits car en 32 pas assez de bits dispo pour MSR)

CR4 → **Supervisor Mode Access Protection**: empêche ring 0 d'accéder à des pages R1,2,3

↳ **Supervisor Mode Execution Protection**: empêche ring 0 d'exécuter à des pages R1,2,3

Control-flow Enforcement Technology:

↳ **protection contre le ROP/JOP**

↳ **shadow stack**: pile dédiée aux transferts de contrôle (@ de retour)

↳ dissocié de la pile de base. CALL sauvegarde l'@retour sur les 2 piles, RET récupère sur les 2 piles et compare les valeurs → si différentes: #CP
 ↳ shadow stack related fault

↳ **indirect branch tracking** ≈ "switch case" → jmp [eax], call *ecx

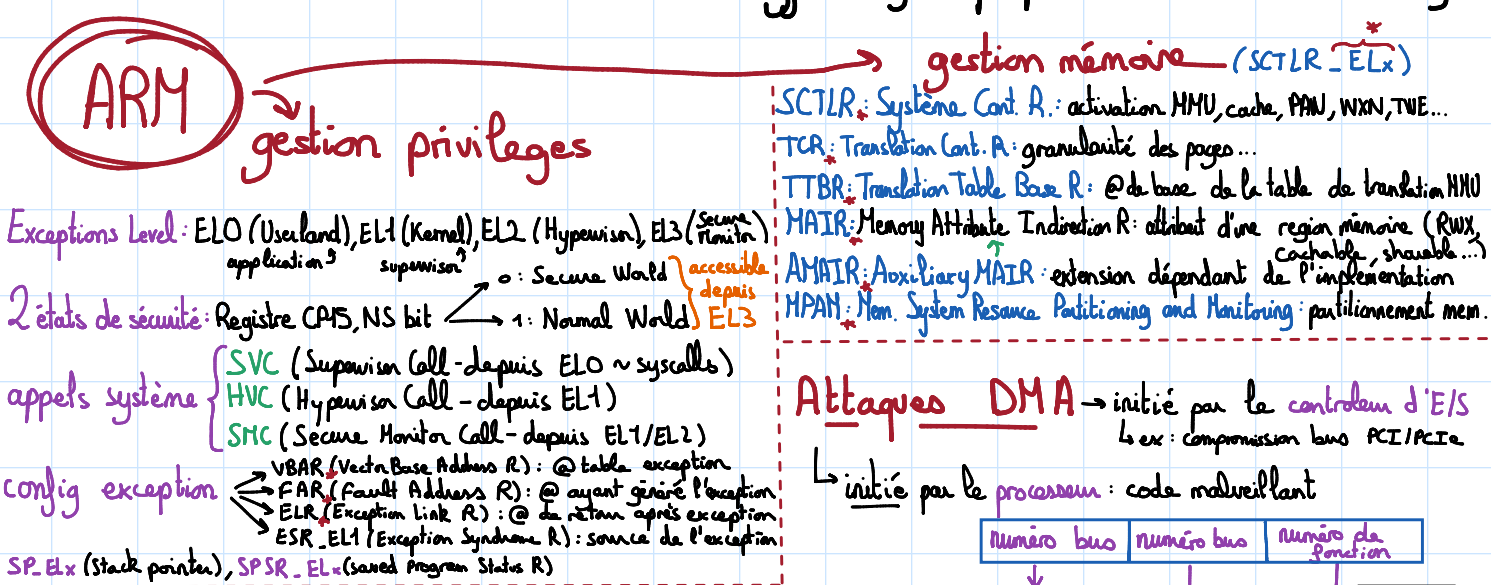
↳ nouvelle instruction ENDBR. à chaque saut indirect: le CPU passe de IDLE à WAIT.FOR.ENDBR, si prochaine instruction pas ENDBR → #CP, sinon IDLE.

↳ cassé par Plundervolt, CrossTalk, attaques cache CPU, bugs synchro enclave
SGX: enclave chiffrée → nécessite un BIOS avec support SGX.

↳ le CPU est le seul environnement où les données sont en clair (ie chiffré en RAM/bus)

↳ seule l'enclave elle-même accède, même pas l'hyperviseur, ni le noyau!

Attaque protection intel: contournement **SHEP**: ROP pour désactiver CR4.SMEP
 ↳ utiliser un buffer Ring 0 qui pointe vers des données Ring 3



IO/MMU → MMU des I/O (dans le northbridge)

↳ Utilise le **DRHU** (DMA Remapping Hardware Unit) pour contrôler les contrôleurs IO en PCIe)

↳ les périphériques sont identifiés auprès d'une **DRHU** par leur id **BDF** (Bus/Device/Fonction)

↳ l'identifiant **BDF** dans le champ "requester-id" ("source-id" pour VT-d)

↳ pour chaque domaine: la table contient les droits (R,W...)

↳ **Limites**: données (VT-d) en mémoire, "source-id" partagé (usurpation), injection d'interruption MSI

MSI: **Message Signaled Interrupt** → à destination d'@ physique

↳ reçu par **LAPIC** qui les traduit en interruptions et les délivre à la CPU demandée

↳ **champ Vector**: même valeur dans l'IDT, **champ Delivery Mode**: Fixed/Lowest Priority

↳ **attaque par injection de SIPI**: 0x40 dans le champ **Delivery Mode** déclenche une interruption non prévue de démarrage. Le processus secondaire (AP) quitte alors son état d'attente pour exécuter du code placé dans le 1er Mo de la RAM (0xVV000)

↳ **attaque par injection d'appels système**: 0x80 dans le champ **Vector** → injection arbitraire

↳ **attaque par injection d'exception (#AC - Alignment Check)**: l'injection de l'exception #AC force IRET à utiliser RFLAGS:CS au lieu de RIP:CS → détournement du flux d'exécution

↳ **mitigation**: Interrupt Remapping → virtualisation des interruptions → filtre!!