

General purpose registers

EAX	31	0
EBX		
ECX		
EDX		
ESI		
EDI		
ESP		(Top) stack pointer
EBP		(Base) stack pointer

Program status and control register

EFLAGS

Program counter

EIP Instruction pointer

Segment selectors

CS	Code segment sel	CPL
SS	Stack segment sel	
DS	Data segment sel	
ES		
FS		
GS		

15

Index	TI	RPL
TI: 0=GDT, 1=LDT		

Security flags

- CD Cache disable: L1/L2/L3 cache restrictions
- NW Not Write-through: cache restrictions
- WP Write Protect: ring0 cannot write RO pages
- PKE Protection-Key-Enable Bit
- SMAP Enables supervisor-mode access prevention
- SMEP Enables supervisor-mode execution prevention
- UMIP User-Mode Instruction Prevention (#GP if SGDT, SIDT, SLDT, SMSW, STR if CPL=0)
- RPL Requestor Privilege Level
- CPL Current Privilege Level

System registers

	Base	Limit
GDT	Global descriptor table register	
LDT	Local descriptor table register	
IDTR	Interrupt descriptor table register	
TR	Task register	

Control registers

CR0	System control flags	
CR1		
CR2	Page Fault #PF linear addr	
CR3	Page directory base phy addr (PDBR)	
CR4	Extension flags	
CR5	IF (Interrupt accept), IOPL (niveau privilege requis IO), NT	
CR6		
CR7		
CR8	Task Priority Register info	

CR0

31 30 29 28 19 18 17 16

PG CD NW AM WP

15

NE ET TS EM MP PE

CR4

31 23 22 21 20 19 18 17 16

PK SM SM OSX PCI FS

E AP EP SAVE DE GS

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

SM VM UM PC PG MC PA PS TS PV VM

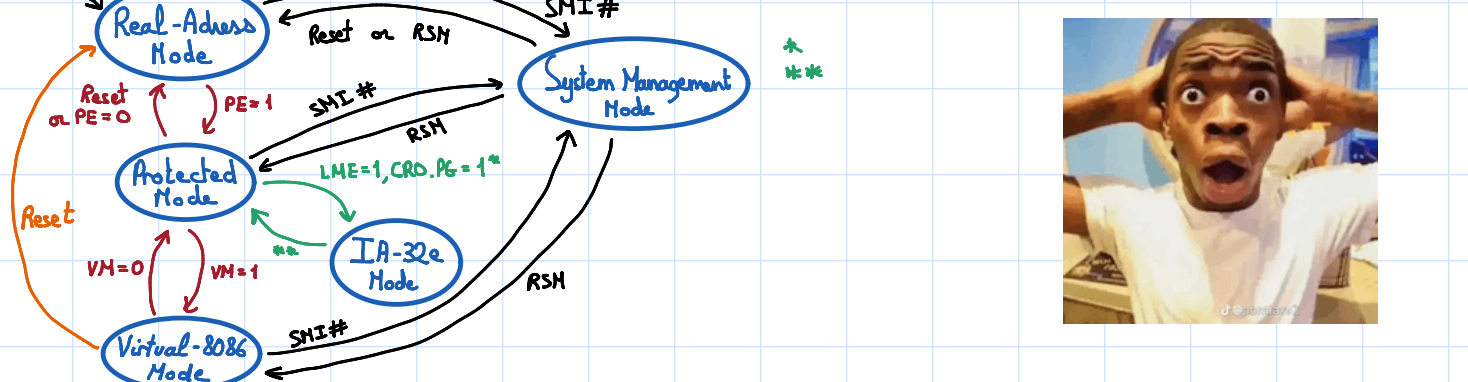
XE XE IP E E E E E DE D I E

Decimal	Type Field					Descriptor Type	Description
	11	10	9	8			
	E	E	W	A			
0	0	0	0	0		Data	Read-Only
1	0	0	0	1		Data	Read-Only, accessed
2	0	0	1	0		Data	Read/Write
3	0	0	1	1		Data	Read/Write, accessed
4	0	1	0	0		Data	Read-Only, expand-down
5	0	1	0	1		Data	Read-Only, expand-down, accessed
6	0	1	1	0		Data	Read/Write, expand-down
7	0	1	1	1		Data	Read/Write, expand-down, accessed
8	1	0	0	0		Code	Execute-Only
9	1	0	0	1		Code	Execute-Only, accessed
10	1	0	1	0		Code	Execute/Read
11	1	0	1	1		Code	Execute/Read, accessed
12	1	1	0	0		Code	Execute-Only, conforming
13	1	1	0	1		Code	Execute-Only, conforming, accessed
14	1	1	1	0		Code	Execute/Read, conforming
15	1	1	1	1		Code	Execute/Read, conforming, accessed

Modes: réel, virtual 8086, system management (SM), protégé, virtualisé (VMX), IA32e (x86-64)

→ 1 seul mode à la fois

→ chaque contexte: qt de mémoire adressable, mode d'usage, gestion interruption/exceptions, instr. non valides/non autorisée



Mode réel: execution 16 bits, @usage 20 bits, pas de niveaux de privilèges

→ registre d'offset sur 16 bits: $2^{16} = 64\text{ kb}$

→ bus d'usage sur 20 bits: $2^{20} = 1\text{ Mb}$

→ usage registres de segments

→ calcul du CPU: @ = selecteur * 16 + offset

→ limites: 1Mb de mémoire, OS gros, multimédia? si l'app plante, tout plante

System Management Mode: mode le plus privilégié, utilisé par le firmware (BIOS), configuré au boot (SHRAN), accessible via System Management Interrupt (SMI), difficilement accessible

VMX: virtualisation matérielle, 2 modes d'exécution: vmx-root pour l'hyperviseur (VMH), vmx-non root pour les VMs

→ filtre les événements systèmes: instruction sensibles, interruptions/exceptions, gestion mémoire, accès aux périphériques

Mode protégé: @usage 16 et 32 bits, émulation du mode réel

→ protégé ⇒ segmentation, pagination, niveaux de privilèges, changement de tâche matériel (task switch)

Espace mémoire en 32 bits → permet une isolation des portions de l'espace linéaire.

→ espace linéaire de 32 bits ⇒ $2^{32} = 4\text{ Gb}$

→ @usage relatif au début d'un segment (@adresse logique)

→ segmentation (m. protégé) → extension du principe de segment du mode R (réel)

→ Modèle flat/protected flat/multi-segment

→ des segments avec des propriétés: type / taille / droits

→ segmentation (m. privilégié)

→ descripteur de segment

→ bases et limites

→ type de segment

→ code (X, RX), data (R, W, RW), système: TSS, Task gate, Interrupt gate, Call gate

31	24	23	22	21	20	19	16	15	14	13	12	11	8	7	0
Base 31:24				G	D	L	A	V	Seg. Limit 19:16	P	D	P	S	Type	Base 23:16
															4

31	16	15	0
Base Address 15:00		Segment Limit 15:00	
		0	

L - 64 bits code segment (IA-32e mode only)

AVL - Available for use by system software

BASE - Segment base address

D/B - Default operation size (0: 16b segment; 1: 32b segment)

DPL - Descriptor Privilege Level

G - Granularity → les limites sont multipliées par la granularité

LIMIT - Segment Limit

P - Segment present

S - Descriptor type (0: system; 1: code or data)

TYPE - Segment type

→ 1 ⇒ limit @ 0x96 ⇒ 4Gb

→ 0 ⇒ 4kb → taille segment défini

Table de descripteurs

→ Global Descriptor Table (GDT), gdtr

→ Local Descriptor Table (LDT), ldtr

→ Interrupt Descriptor Table (IDT), idtr

→ unique pour chaque cœur d'un CPU

→ 8192 entrées max, première entrée vide

→ locale à une tâche

→ son propre descripteur se trouve dans GDT

→ unique pour chaque cœur d'un CPU

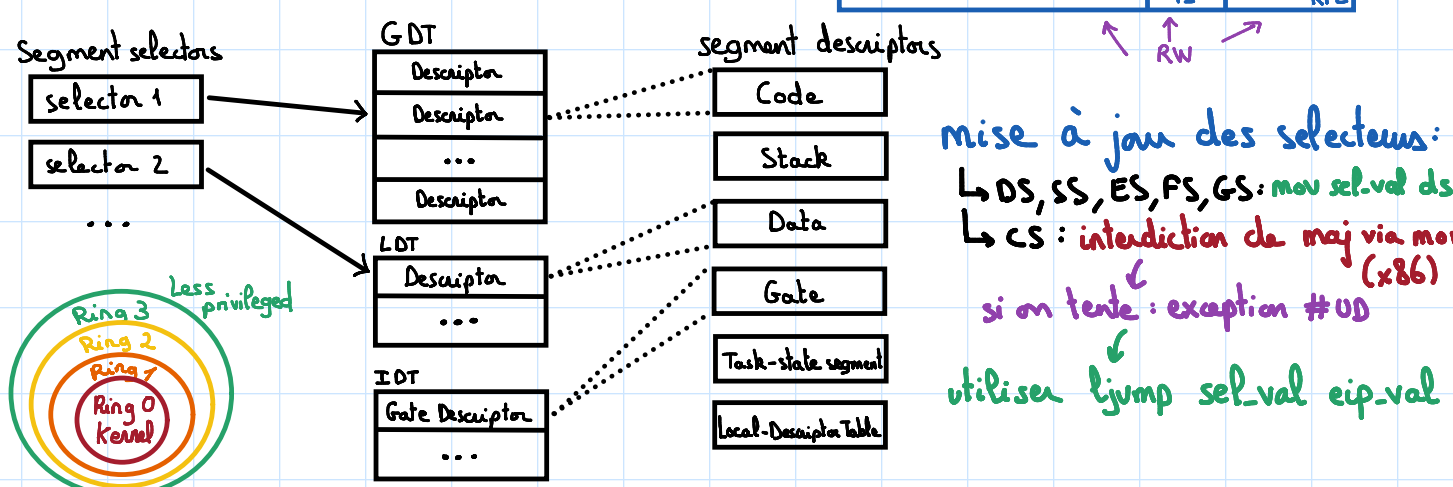
→ utilisée pour les exceptions/interruptions

Selecteur de segment → utilisé par les registres de segment et permet l'accès à un descripteur.

→ chaque selecteur défini: niveau de privilège, une table (local ou globale), un indice de descripteur dans cette table.

→ le CPU charge le descripteur dans des parties cachées du registre

→ partie visible au programmeur



pour jump: saut long, donner selecteur et offset, selecteur dans CS, offset dans EIP

Modèle de segmentation: modèle "Multi-segments", modèle "FLAT"

Niveaux de privilèges: segments disjointes

→ prend toute la mémoire, les descripteurs pointent au même endroit

→ CPL: Current Privilege Level (dans cs)

→ RPL: Requestor Privilege Level (au niveau du selecteur)

→ DPL: Descriptor Privilege Level (au niveau du descripteur)

→ complexe

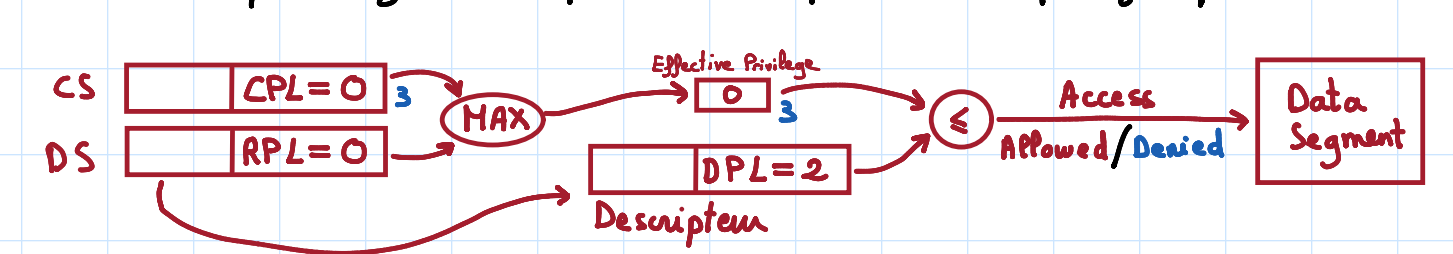
→ interprétation différente selon le type de segment

→ transients inter-segments (Gate)

→ changement du niveau de privilèges → call gate ou interrupt gate

→ on ne peut charger un descripteur de code qu'à niveau de privilège équivalent

typical: appel système (ring 3 vers 0)



Aujourd'hui: pagination au lieu de segmentation!

Exceptions: générées par le CPU, en cas d'erreurs (#GP, #NP) ou par du code (#BP)

→ différentes natures:

- fault: on la gère et on re-exécute l'instruction
- trap: on la gère et on continue après l'instruction
- abort: non récupérable

Interruptions: générées par les périphériques ou du code (0x80)

→ irq (pour les périphériques), smi (pour le mode SM), nmi (par des périph. liés à l'ACPI), ipi (entre cœurs d'un CPU, entre CPUs (smp))

→ masquables via l'instruction cli ⇒ CPU in-interruptible, sauf nmi (Non Maskable Interrupt)

General Protection Fault (#GP): fournit le selecteur de segment fautive (si seg fault: taille, droit, pte)

Break Point (#BP): type trap, pas de code d'erreur, survient lorsque int3 est exécuté

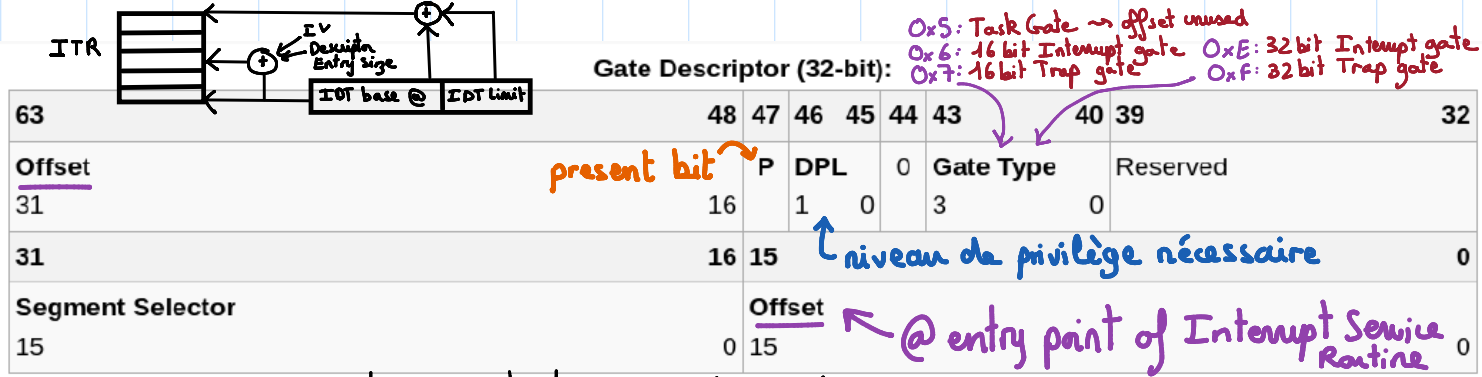
Invalid Opcode (#UD); Divide Error (#DE): type fault, pas de code d'erreur

Page Fault (#PG): page non présente (pte.p / pte.p=0), accès R/W, accès user/supervisor

→ CR2 contient l'adresse fautive

→ Instruction privilégiée: LGDT, LIDT, LTR, MOV CRx, HLT, INVLPG, STI, CLI ⇒ #GP

→ si ≠ Ring 3



changement de contexte: niveau de privilège, valeur registres g. et registres états

↳ le CPU sauvegarde pour nous les infos du niveau de provenance (la pousse le registre)

3 étapes: CPU sauvegarde et check si chgt de privilège, sauvegarde CTX, iret



Appels systèmes: historiquement -> interruption ring 3, gate DPL=3, int N

↳ aujourd'hui: fast system call (sysenter, sysexit), l'OS configure des MSRs

↳ Ring 3: int 0x80 -> CPU: { esp = TSS.esp0, eip = IDT[0x80].offset + GDT[IDT[0x80].selector].base -> Ring 0: { syscall_exp_handler: syscall_table[aux] }

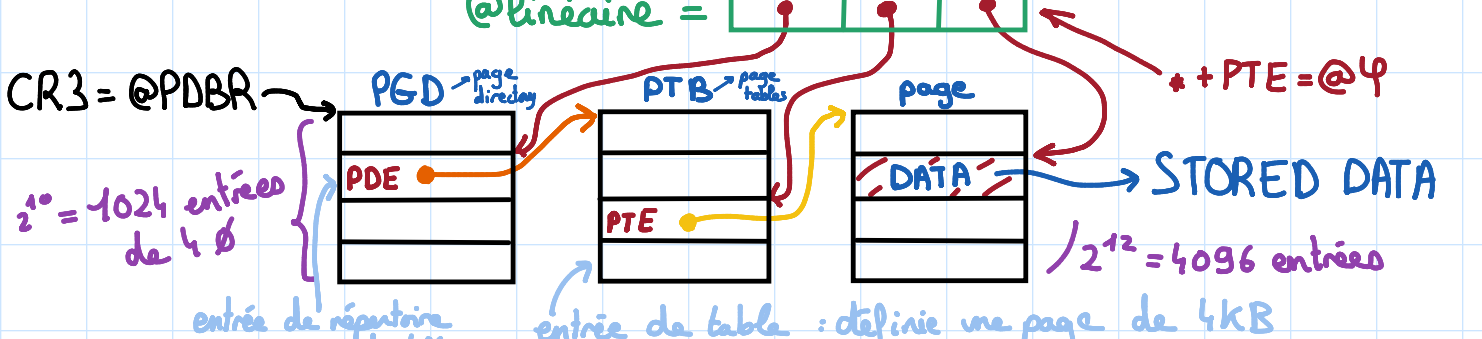
Pagination: traduction @physique/@virtuelle par MMU (Memory Management Unit)

↳ optimisation conso mémoire: demand paging (ie memory), shared memory (ie thread)

↳ charger même programme plusieurs fois: m @virtuelles, ≠ @physiques

• @logique + offset = @linéaire } **segmentation** ~ offset = 0

@logique = @linéaire



→ 1 table de page couvre 1024 x 4Ko = 4Mo d'espace virtuel

→ 1 répertoire de page couvre donc 1024 x 4Mo = 4GB d'espace virtuel

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	PTE, PDE
Address of page directory ¹																Ignored				P C D	P W T	Ignored		CR3								
Bits 31:22 of address of 4MB page frame								Reserved (must be 0)				Bits 39:32 of address ²				P A T	Ignored	G	1	D	A	P C D	P W T	U / S	R / W	1	PDE: 4MB page					
Address of page table																Ignored				Q	I g n	A	P C D	P W T	U / S	R / W	1	PDE: page table				
Ignored																Ignored				Global ? (mapping pouris tant)				Q	PDE: not present							
Address of 4KB page frame																Ignored				G	P A T	D	A	P C D	P W T	U / S	R / W	1	PTE: 4KB page			
Ignored																bits de gestion cache				user (ring 3) supervisor (r-0, 1, 2)				Q	PTE: not present							

Mode		Physical-Address Extensions (CR4.PAE)	Page-Size Extensions (CR4.PSE)	Page-Directory Pointer Offset	Page-Directory Page Size	Resulting Physical-Page Size	Maximum Virtual Address	Maximum Physical Address
Long Mode	64-Bit Mode	Enabled	—	PDPE.PS=0	PDE.PS=0	4 Kbyte	64-bit	52-bit
	PDE.PS=1				2 Mbyte			
	Compatibility Mode			PDPE.PS=1	—	1 Gbyte		
Legacy Mode		Enabled	—	PDPE.PS=0	PDE.PS=0	4 Kbyte	32-bit	52-bit
					PDE.PS=1	2 Mbyte		52-bit
		Disabled	Disabled		—	4 Kbyte		32-bit
			Enabled		PDE.PS=0	4 Kbyte		32-bit
					PDE.PS=1	4 Mbyte		40-bit

Translation Lookaside Buffers (TLBs) ~ cache traduct @virt/@p

↳ écriture dans CR3 => TLB flush (invalide la TLB)

↳ entrées global conservées (sauf si écriture CR4)

Problème? toutes les tables contiennent des @p -> comment les lire/modifier si pagination activée (normalement @virt)

↳ identity mapping (page), PAGE-OFFSET (linéaire/p), self-mapping?

SELF-MAPPING:

~ annulation 1^{ère} indirection

~ les pages finales associées sont les tables de pages

~ inconvénient: on perd 4Mo d'espace virtuel

fonctionnalités attendues d'un OS: isolation, limitation, gestion erreurs

↳ en vrac: auth., data execution protection, ASLR, intégrité, sandboxing

Accès données (DS, ES, ...) si $\max(CPL, RPL) \leq DPL \Rightarrow RPL \geq CPL$

Accès code (JMP, CALL) si $CPL == DPL$

Accès stack si $CPL == DPL$

NX/DEP ~ no execute (pas X dans stack/heap) ^{BoF}

ANNALES & RÉPONSES

2024 Q7 : Processus Ring 3 tente instruction Ring 0 ? R : L'instruction est refusée, une exception #GP est levée.

ANNALES & RÉPONSES

2024 Q13 : Quoi mettre sur la pile avant IRET vers Ring 3 ? R : Il faut simuler le contexte empilé par une interruption : SS, ESP, EFLAGS, CS, EIP.

ANNALES & RÉPONSES

2024 : Sécurité Segmentation Flat Model ? R : Utilité faible pour l'isolation mémoire (tout le monde voit 0-4Go). Sert uniquement à gérer les Privilèges (DPL 0 vs 3).

ANNALES & RÉPONSES

2024 : Identity Mapping & Sécurité ? R : Si mappé Supervisor (U/S=0) : OK (Noyau protégé). Si mappé User (U/S=1) sur zone noyau : FAILLE (User peut lire/écrire le noyau).

ANNALES & RÉPONSES

2024 : Où est l'adresse de base du segment ? R : Spécifiée dans le Descripteur de Segment (pointé par le sélecteur). Pas dans GDTR (qui pointe la table).

ANNALES & RÉPONSES

2023 : Qui configure la GDT ? R : Le Noyau (logiciel) écrit la GDT en RAM. Le CPU (matériel) la lit via GDTR pour vérifier les accès.

ANNALES & RÉPONSES

Interruption : Rôle CPU vs Noyau (Sauvegarde) R : CPU (Hardware) : Sauve l'état critique (CS, EIP, EFLAGS + SS/ESP si chgt Ring). Met EIP = Handler. Noyau (Software) : Sauve le reste (Généraux) via PUSHA.

ANNALES & RÉPONSES

2019 : Danger de sauver le contexte en mémoire User ? R : Si le noyau sauve EIP/ESP dans la mémoire de la tâche, la tâche peut modifier ces valeurs. Le contexte doit être sauvé sur la Pile Noyau.

ANNALES & RÉPONSES

2019 : Différence privilèges Segmentation vs Pagination ? R : Segmentation = 4 niveaux (0-3). Pagination = 2 niveaux (User/Supervisor). Ring 0,1,2 = Supervisor.

Checklist d'Audit (A appliquer dans l'ordre)

- Segmentation (RPL/DPL) :**
 - Regarder le sélecteur CS de la tâche.
 - Si RPL (2 derniers bits) = 0 -> Ring 0 (Root). Danger!
 - Vérifier si le segment de code est Conforming (Rare, mais permet CPL < DPL).
- Pagination (Droits U/S & R/W) :**
 - Regarder l'adresse virtuelle cible (ex : Code Noyau, Table des Processus).
 - Trouver les PDE/PTE correspondants.
 - Si bit U/S = 1 (User) pour une zone Noyau -> FAIL. L'utilisateur peut lire/écrire.
 - Si bit R/W = 1 sur zone Code -> Risque d'injection de code (Self-Modifying).
- Appels Syst'eme (Entrées) :**
 - Bounds Check : 'if (ptr < LIMITE)' ne suffit pas! Vérifier si 'ptr' pointe vers une zone sensible (IDT, GDT, Code Syscall) qui serait sous la limite.
 - Pointeurs User : Le noyau doit vérifier que les pointeurs fournis par l'user pointent bien en zone User ('access_ok'). Sinon -> Write-What-Where.

Scénario 1 : Write-What-Where (Classique)

Code : sys_write(int* ptr, int val) { *ptr = val; } Faille : Pas de vérification de 'ptr'. Attaque : 1. 'ptr' = Adresse IDT (Entrée 0x80). 2. 'val' = Adresse Shellcode User. 3. Appeler 'int 0x80' -> Exécution Shellcode en Ring 0.

Scénario 2 : Bypass Vérification (Annales 2024)

Code : if (addr <= 0x300000 || addr >= 0xFFFF0000) return -1; Config : Handler Syscall à 0x00340000. Faille : L'adresse cible (0x340000) est > 0x300000 et < 0xFFFF.... Le check passe! Impact : On peut écraser le code du Handler Syscall (même si U/S=0, car c'est le noyau qui écrit pour nous).

Scénario 3 : Segmentation Malveillante (Annales 2023)

Cas : Une appli A2 est lancée avec le sélecteur 0x08 (Index 1, RPL 0). Analyse : RPL=0 -> L'appli tourne en Ring 0. Elle a accès total (IO, CR3, HLT...). Cas : Appli A3 (Ring 3) veut appeler int 0x80. Règle : Il faut DPL(Gate) >= CPL(3). Sinon #GP. (C'est la porte qui doit être accessible, pas le code derrière).

Scénario 4 : Identity Mapping (Annales 2019/24)

Config : Le noyau mappe la RAM physique (0-X Mo) à l'adresse virtuelle 0. Risque 1 : Si U/S=1 -> User lit/modifie tout le noyau. Risque 2 (Subtil) : Facilite les exploits (KASLR inefficace car on connaît les adresses physiques des structures clés).

Scénario 5 : Modification Tables (Annales 2019)

But : Le noyau veut changer le mapping d'une page. Problème : Il connaît l'adresse Physique de la PT (via PDE), mais pour écrire dedans ('MOV [addr], val'), il lui faut une adresse Virtuelle. Solution : Il doit avoir mappé cette page physique quelque part (Self-Mapping ou Identity Mapping). Sinon, il ne peut pas la modifier.