EDR et XDR Bypass 2026 : Techniques Offensives et Détection

Le contournement des solutions EDR (Endpoint Detection and Response) et XDR (Extended Detection and Response) est devenu en 2026 l'une des compétences les plus critiques à maîtriser pour les équipes Red Team et, en miroir, l'un des défis les plus complexes à relever pour les équipes de défense. Les EDR modernes — CrowdStrike Falcon, Microsoft Defender for Endpoint, SentinelOne, Elastic Security — ne se contentent plus de signatures statiques : ils analysent les comportements en temps réel, instrumentent le kernel, surveillent les appels système et corrèlent les événements à l'échelle du parc via des moteurs de threat intelligence. Face à cette sophistication défensive, les attaquants ont développé un arsenal de techniques de bypass de plus en plus élaborées : process hollowing, PPID spoofing, Heaven's Gate pour l'accès aux syscalls natifs, BYOVD (Bring Your Own Vulnerable Driver) pour le patching kernel, ETW patching pour aveugler la télémétrie, et AMSI bypass pour neutraliser l'analyse des scripts PowerShell et .NET. Ce guide technique approfondi présente les principales techniques offensives de bypass EDR/XDR 2026, leur fonctionnement interne, les indicateurs de détection associés, et les contre-mesures défensives que les équipes Blue Team peuvent déployer. L'objectif n'est pas de fournir un kit d'exploitation clé en main, mais de donner aux professionnels de la sécurité offensive et défensive la compréhension technique nécessaire pour tester et renforcer leurs défenses de manière éclairée et conforme aux cadres réglementaires en vigueur.

Résumé exécutif

Contexte 2026 : les EDR/XDR de nouvelle génération déplacent la détection vers le comportement runtime — les bypass modernes doivent donc cibler les hooks, la télémétrie et le kernel lui-même

Techniques offensives principales : Process Hollowing, PPID Spoofing, Heaven's Gate (WoW64 syscalls), ETW/AMSI patching, BYOVD, Kernel Callback removal

Détection comportementale : les EDR détectent par API hooking (user-mode), kernel callbacks (ETW-TI, PsSetCreateProcessNotifyRoutine), et télémétrie réseau (XDR)

MITRE ATT&CK : T1055 (Process Injection), T1036 (Masquerading), T1562.001 (Impair Defenses), T1543 (Create or Modify System Process)

Contre-mesures défensives : Microsoft Vulnerable Driver Blocklist, Kernel Patch Protection, System Guard Secure Launch, EDR tamper protection

Cadre légal : toute utilisation de ces techniques sans autorisation écrite est une infraction pénale (articles 323-1 et suivants du Code pénal)

Comment fonctionnent les EDR modernes : architecture de détection

Pour comprendre les techniques de bypass, il faut d'abord saisir comment les EDR interceptent les activités malveillantes. Les solutions modernes combinent plusieurs couches de détection complémentaires :

Hooking user-mode via l'IAT et les inline hooks

La majorité des EDR injectent une DLL dans chaque processus (souvent via un pilote kernel) et posent des hooks sur les fonctions critiques de ntdll.dll et kernel32.dll. Ces hooks redirigent les appels vers des fonctions d'analyse EDR avant de transmettre l'appel original. Les fonctions les plus hookées incluent : NtCreateThread, NtAllocateVirtualMemory, NtWriteVirtualMemory, NtCreateProcess, NtOpenProcess.

Callbacks kernel via ETW-TI et PatchGuard

Au niveau kernel, les EDR s'appuient sur des callbacks Windows documentés : PsSetCreateProcessNotifyRoutineEx (création de processus), PsSetLoadImageNotifyRoutine (chargement de DLLs), ObRegisterCallbacks (accès aux handles). ETW-TI (Event Tracing for Windows — Threat Intelligence) est une fonctionnalité kernel qui fournit aux EDR une télémétrie privilégiée non patchable depuis le user-mode.

Télémétrie réseau et corrélation XDR

Les XDR ajoutent une couche de corrélation multi-sources : endpoint + réseau + email + identité. Un processus qui injecte du code ET qui établit une connexion C2 immédiatement après sera détecté même si l'injection individuelle passe sous les radars. La corrélation temporelle et comportementale est le principal avantage des XDR sur les EDR purs.

Process Hollowing : injection via le remplacement de processus légitime

Le **Process Hollowing** (ou Process Replacement) est une technique classique de process injection qui permet d'exécuter du code malveillant sous l'apparence d'un processus légitime (svchost.exe, explorer.exe, etc.). La séquence typique :

```
 // Séquence Process Hollowing (C/C++ - à des fins éducatives uniquement)
// 1. Créer le processus cible en mode SUSPENDED
CreateProcess("C:\\Windows\\System32\\svchost.exe", NULL, NULL, NULL,
              FALSE, CREATE_SUSPENDED, NULL, NULL, &si, &pi);

// 2. Unmapper la mémoire originale du processus
NtUnmapViewOfSection(pi.hProcess, pImageBase);

// 3. Allouer de la mémoire à la même adresse (ou autre adresse si ASLR)
NtAllocateVirtualMemory(pi.hProcess, &pImageBase, 0, &dwSize,
                        MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);

// 4. Écrire le PE malveillant (shellcode ou PE complet)
WriteProcessMemory(pi.hProcess, pImageBase, pPayload, dwPayloadSize, NULL);

// 5. Modifier le contexte du thread pour pointer vers le nouveau point d'entrée
SetThreadContext(pi.hThread, &ctx);

// 6. Reprendre le thread
ResumeThread(pi.hThread);
```

Détection EDR : Les hooks sur NtUnmapViewOfSection et NtWriteVirtualMemory, combinés à la détection de la divergence entre l'image on-disk et l'image en mémoire (technique dite "image path mismatch"), permettent aux EDR de détecter le Process Hollowing avec un faible taux de faux positifs. SentinelOne et CrowdStrike détectent spécifiquement la séquence "CreateSuspended — UnmapView — WriteProcessMemory — ResumeThread".

PPID Spoofing : masquer l'arbre de processus

Le **PPID Spoofing** (Parent Process ID Spoofing) permet de créer un processus en lui attribuant un faux processus parent, afin de masquer la chaîne d'exécution. Un processus malveillant qui se présente comme fils de svchost.exe ou explorer.exe paraît moins suspect qu'un fils de cmd.exe ou powershell.exe.

```
 // PPID Spoofing via InitializeProcThreadAttributeList
HANDLE hParent = OpenProcess(PROCESS_CREATE_PROCESS, FALSE, targetPPID);

SIZE_T size = 0;
InitializeProcThreadAttributeList(NULL, 1, 0, &size);
LPPROC_THREAD_ATTRIBUTE_LIST pAttrList = HeapAlloc(GetProcessHeap(), 0, size);
InitializeProcThreadAttributeList(pAttrList, 1, 0, &size);

UpdateProcThreadAttribute(pAttrList, 0,
    PROC_THREAD_ATTRIBUTE_PARENT_PROCESS, &hParent,
    sizeof(HANDLE), NULL, NULL);

STARTUPINFOEX sie = { sizeof(sie) };
sie.lpAttributeList = pAttrList;

CreateProcess("C:\\Windows\\System32\\notepad.exe", NULL, NULL, NULL,
              FALSE, EXTENDED_STARTUPINFO_PRESENT, NULL, NULL,
              (LPSTARTUPINFOA)&sie, &pi);
```

Détection : Les EDR modernes comparent le PPID déclaré avec la réalité de la chaîne d'héritage. Un processus notepad.exe fils de svchost.exe (PID 4) est statistiquement anormal. Des outils comme Sysmon (EventID 1) capturent le ParentProcessId et permettent des règles SIGMA précises.

Heaven's Gate : appels système 64 bits depuis un processus 32 bits

Heaven's Gate est une technique qui exploite la couche de compatibilité WoW64 (Windows 32-bit on 64-bit) pour effectuer des appels système 64 bits directement depuis un processus 32 bits, contournant ainsi les hooks user-mode des EDR qui ne surveillent que le segment 32 bits de la mémoire du processus. En passant en mode 64 bits (via un far jump vers le segment 0x33), le code peut appeler directement les syscalls ntoskrnl.exe natifs.

Les EDR qui déploient des agents 32 bits uniquement (ou qui ne gèrent pas correctement le passage 32/64 bits) sont aveuglés. Les solutions modernes comme Falcon ou Defender for Endpoint déploient des hooks 64 bits et des callbacks kernel insensibles à Heaven's Gate.

ETW Patching : neutraliser la télémétrie Windows

ETW (Event Tracing for Windows) est le mécanisme kernel qui fournit aux EDR une grande partie de leur télémétrie. Un attaquant peut patcher la fonction EtwEventWrite dans ntdll.dll (user-mode) pour que tous les événements ETW générés depuis le processus courant soient silencieusement supprimés.

```
 // ETW Patching - neutraliser EtwEventWrite dans le processus courant
// AVERTISSEMENT : technique illégale hors cadre Red Team autorisé
ULONG_PTR pEtwEventWrite = (ULONG_PTR)GetProcAddress(
    GetModuleHandleA("ntdll.dll"), "EtwEventWrite");

DWORD dwOldProtect;
VirtualProtect((LPVOID)pEtwEventWrite, 1, PAGE_EXECUTE_READWRITE, &dwOldProtect);
*(BYTE*)pEtwEventWrite = 0xC3; // RET instruction = NOP effectif
VirtualProtect((LPVOID)pEtwEventWrite, 1, dwOldProtect, &dwOldProtect);
```

Contre-mesure défensive : ETW-TI (ETW Threat Intelligence Provider) opère au niveau kernel et ne peut pas être patché par ce mécanisme user-mode. De plus, les EDR modernes surveillent les modifications de ntdll.dll en mémoire et alertent sur tout patch de fonctions critiques. Microsoft Defender for Endpoint détecte spécifiquement le patching de EtwEventWrite.

AMSI Bypass : neutraliser l'analyse des scripts

AMSI (Antimalware Scan Interface) est une API Windows qui permet aux solutions de sécurité d'analyser le contenu des scripts PowerShell, VBScript, JScript et des assemblies .NET avant leur exécution. Contourner AMSI est souvent la première étape d'une intrusion via scripts.

```
 # AMSI Bypass - compréhension défensive uniquement
# Le pattern classique cherche à patcher AmsiScanBuffer dans amsi.dll
# Les EDR modernes détectent :
# - Les appels GetProcAddress("amsi.dll", "AmsiScanBuffer")
# - Les modifications mémoire de la DLL amsi.dll
# - Les patterns reflection .NET pour charger du code sans AMSI
# - Script Block Logging PowerShell (EventID 4104) capture les scripts déobfusqués

# Activation du Script Block Logging (défensif)
Set-ItemProperty HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging `
    -Name EnableScriptBlockLogging -Value 1
```

Les EDR de 2026 détectent les AMSI bypass via plusieurs vecteurs : surveillance des appels LoadLibrary sur amsi.dll, détection des modifications mémoire de AmsiScanBuffer, Script Block Logging PowerShell (EventID 4104), et analyse comportementale des patterns de reflection .NET.

BYOVD : Bring Your Own Vulnerable Driver

La technique **BYOVD** consiste à charger un pilote Windows légitime mais vulnérable pour obtenir une exécution arbitraire en ring 0 (kernel mode) et désactiver les protections EDR depuis le kernel. Cette technique est particulièrement redoutée car elle contourne PatchGuard (KPP) en utilisant du code signé Microsoft ou par un éditeur de confiance.

Mécanisme d'attaque BYOVD

Étape 1 : Charger un pilote vulnérable signé (ex: gdrv.sys de Gigabyte, RTCore64.sys d'MSI, mhyprot2.sys d'anti-cheat)

Étape 2 : Exploiter la vulnérabilité du pilote pour écrire en mémoire kernel arbitrairement (IOCTL malveillant)

Étape 3 : Supprimer les kernel callbacks de l'EDR (PsSetCreateProcessNotifyRoutineEx, ObRegisterCallbacks)

Étape 4 : Désactiver le processus EDR en le killant depuis le kernel ou en supprimant sa protection

Exemples réels 2024-2026 : les groupes Lazarus (Corée du Nord) utilisent BYOVD dans leurs campagnes financières ; BlackCat/ALPHV a utilisé mhyprot2.sys pour contourner les EDR lors de déploiements ransomware.

Contre-mesures BYOVD

Microsoft Vulnerable Driver Blocklist (WDAC) : liste de pilotes bloqués maintenue par Microsoft, intégrée dans Windows Defender Application Control. Mise à jour régulière recommandée.

HVCI (Hypervisor-Protected Code Integrity) : empêche le chargement de code kernel non signé et limite la portée des BYOVD même avec un pilote vulnérable signé

Secure Boot + Kernel DMA Protection : protège contre les vecteurs hardware (Thunderbolt, DMA attacks)

EDR tamper protection : les solutions comme CrowdStrike Falcon implémentent des protections anti-kill via PPL (Protected Process Light)

```
 # Vérifier l'état HVCI (Hypervisor Protected Code Integrity)
Get-CimInstance -ClassName Win32_DeviceGuard -Namespace root\Microsoft\Windows\DeviceGuard |
    Select-Object VirtualizationBasedSecurityStatus, CodeIntegrityPolicyEnforcementStatus

# Activer la Vulnerable Driver Blocklist via registre
# HKLM\SYSTEM\CurrentControlSet\Control\CI\Config\VulnerableDriverBlocklistEnable = 1

# Sysmon EventID 6 : chargement de pilote kernel - règle de détection BYOVD
# Alerte si ImageLoaded correspond à la liste de pilotes vulnérables connus
<RuleGroup name="BYOVD Detection" groupRelation="or">
  <DriverLoad onmatch="include">
    <ImageLoaded condition="contains">gdrv.sys</ImageLoaded>
    <ImageLoaded condition="contains">mhyprot</ImageLoaded>
    <ImageLoaded condition="contains">RTCore64</ImageLoaded>
  </DriverLoad>
</RuleGroup>
```

Kernel Callback Removal : désactiver l'EDR depuis le kernel

Une fois l'accès kernel obtenu (via BYOVD ou une vulnérabilité kernel), un attaquant peut supprimer directement les callbacks que l'EDR a enregistrés. Les callbacks PsSetCreateProcessNotifyRoutineEx, PsSetCreateThreadNotifyRoutine et ObRegisterCallbacks sont stockés dans des tables kernel accessibles. En supprimant les entrées correspondant au driver EDR, l'attaquant aveugle complètement la solution.

Détection par les XDR : Les solutions XDR avancées comme Microsoft Defender for Endpoint (avec Microsoft Sentinel) peuvent détecter la suppression de callbacks via la télémétrie Kernel Notify Routine Removal. Des règles KQL permettent d'alerter sur ce pattern :

```
 // KQL - Détection de suppression de kernel callbacks (Microsoft Sentinel)
DeviceEvents
| where ActionType == "KernelNotifyRoutineRemoval"
| where InitiatingProcessFileName !in ("System", "wininit.exe")
| project TimeGenerated, DeviceName, InitiatingProcessFileName,
          InitiatingProcessCommandLine, AdditionalFields
| order by TimeGenerated desc
```

MITRE ATT&CK : mapping des techniques EDR bypass

Technique MITRE	ID	Description	Bypass associé
Process Injection	T1055	Injection de code dans un processus légitime	Process Hollowing, DLL Injection
Masquerading	T1036.005	Match Legitimate Name or Location	PPID Spoofing, binary renaming
Impair Defenses	T1562.001	Disable or Modify Tools	ETW patching, AMSI bypass, BYOVD
Rootkit	T1014	Kernel-level hide from detection	BYOVD + callback removal
Virtualization Evasion	T1497	Sandbox detection	Anti-debugging, timing checks
System Binary Proxy	T1218	Exécution via binaire Windows légitime	LOLBins (rundll32, mshta, certutil)
OS Credential Dumping	T1003	Extraction de credentials	Post-bypass : Mimikatz, Nanodump

Stratégie défensive globale contre les bypass EDR/XDR

Les équipes Blue Team peuvent s'appuyer sur les ressources suivantes pour renforcer leur défense contre les bypass. L'intégration avec des outils comme Cobalt Strike dans les exercices Red Team permet de valider l'efficacité des détections. Des tests réguliers avec des outils de test d'intrusion complémentaires permettent de couvrir les vecteurs web et réseau en parallèle. L'analyse des chemins d'attaque AD avec BloodHound permet d'identifier comment un attaquant post-bypass chercherait à pivoter dans l'AD. Pour aller plus loin sur les techniques d'intrusion avancées, notre guide sur les attaques NTLM Relay 2026 couvre les vecteurs réseau complémentaires aux bypass EDR.

Les fiches MITRE ATT&CK T1562.001 fournissent des procédures de test et des données d'adversaires réels pour calibrer les règles de détection. La guidance CISA sur l'arrêt des ransomwares intègre des recommandations spécifiques pour la protection des EDR contre les tentatives de désactivation.

Mesures défensives prioritaires contre les bypass EDR/XDR

Activer HVCI sur tous les postes Windows 10/11 — bloque la majorité des BYOVD

Maintenir à jour la Vulnerable Driver Blocklist via Windows Update et WDAC

PPL (Protected Process Light) pour les processus EDR — empêche leur kill depuis le user-mode

Script Block Logging PowerShell (EventID 4104) + module logging — détecte les AMSI bypass dans les scripts

ETW-TI provider activé — télémétrie kernel non neutralisable depuis le user-mode

Sysmon avec règles SIGMA — détection de Process Hollowing, PPID Spoofing, driver loading

Tamper Protection activée dans Windows Security — empêche la modification des settings Defender

Exercices Red Team réguliers avec émulation d'adversaires (MITRE ATT&CK) pour valider les détections

À RETENIR

Points clés à retenir

Les **EDR/XDR modernes** détectent par comportement runtime — les bypass doivent cibler les hooks, la télémétrie ETW et les callbacks kernel

Process Hollowing et PPID Spoofing sont détectés par les EDR via les divergences image/mémoire et les arbres de processus anormaux

BYOVD est la menace la plus sérieuse : elle opère depuis le kernel et peut désactiver complètement un EDR — HVCI est la contre-mesure principale

ETW Patching user-mode ne contourne pas ETW-TI (kernel-mode) — les EDR qui utilisent ce provider restent efficaces

L'AMSI peut être contourné mais sa désactivation est elle-même un signal de détection pour les EDR modernes

Les XDR ajoutent la corrélation multi-source : un bypass isolé peut passer, la combinaison bypass + C2 + mouvement latéral ne passe pas

Cadre légal : ces techniques ne s'utilisent que dans le cadre d'un mandat de pentest signé ou d'un exercice Red Team avec autorisation formelle

FAQ — EDR et XDR Bypass 2026

Quelle est la différence entre un EDR et un XDR pour résister aux techniques de bypass ?

Un EDR (Endpoint Detection and Response) surveille et répond aux menaces sur un endpoint individuel via des agents kernel, des hooks user-mode et une analyse comportementale locale. Un XDR (Extended Detection and Response) étend cette couverture à l'ensemble de l'infrastructure — réseau, email, identité, cloud — et corrèle les signaux de multiples sources. Face aux bypass techniques, un EDR seul peut être aveuglé par des techniques comme le BYOVD (qui désactive l'agent kernel) ou l'ETW patching. Un XDR, lui, continuera à voir les activités réseau du C2, les mouvements sur les logs d'identité (Active Directory), et les anomalies comportementales même si l'agent endpoint est compromis. En 2026, les plateformes XDR comme Microsoft Defender for Endpoint + Microsoft Sentinel ou CrowdStrike Falcon + LogScale offrent une résilience nettement supérieure aux techniques de bypass pures.

Comment les Red Teams testent-elles légalement les bypass EDR en entreprise ?

La procédure légale et éthique passe par plusieurs étapes obligatoires : un contrat de prestation Red Team signé définissant précisément le périmètre autorisé, les techniques autorisées et les systèmes cibles ; une Rules of Engagement (RoE) validée par le RSSI et la direction ; une salle de guerre (war room) avec les équipes Blue Team qui peuvent suivre ou arrêter l'exercice ; un rapport post-exercice couvrant les techniques utilisées, les détections réussies et manquées, et les

recommandations. Les outils de simulation d'adversaires comme Atomic Red Team (MITRE), Caldera ou les modules Cobalt Strike permettent de tester des techniques de bypass dans un cadre contrôlé.

Heaven's Gate est-il encore pertinent contre les EDR 2026 ?

Heaven's Gate est une technique de plus en plus désuète contre les EDR modernes pour plusieurs raisons. Premièrement, les solutions de 2026 déploient systématiquement des agents 64 bits qui surveillent les transitions WoW64 via des callbacks kernel — la transition de segment ne masque plus les activités. Deuxièmement, ETW-TI (Threat Intelligence Provider) capture les événements kernel indépendamment de l'architecture du processus appelant. Troisièmement, les EDR peuvent instrumenter le noyau directement pour surveiller les far jumps 32 vers 64 bits. Heaven's Gate reste pertinent comme composante d'une chaîne d'attaque sophistiquée (multi-stage loader) mais ne constitue plus à lui seul un bypass EDR fiable en 2026.

Comment se protéger contre le BYOVD en pratique ?

La protection contre le BYOVD repose sur plusieurs couches complémentaires. Premièrement, activer HVCI (Hypervisor-Protected Code Integrity) sur tous les postes Windows 10/11 — cette mesure empêche l'exécution de code kernel non signé même via un pilote vulnérable. Deuxièmement, maintenir à jour la Microsoft Vulnerable Driver Blocklist via WDAC (Windows Defender Application Control) — Microsoft publie des mises à jour régulières intégrant les nouveaux pilotes vulnérables découverts. Troisièmement, configurer des règles Sysmon (EventID 6) alertant sur le chargement de pilotes connus comme vulnérables. Quatrièmement, implémenter Windows Defender Application Control (WDAC) en mode Allowlist pour interdire le chargement de tout pilote non explicitement autorisé.

Direct Syscalls et Indirect Syscalls : contourner les hooks user-mode

Les **Direct Syscalls** constituent une approche de plus en plus populaire pour contourner les hooks EDR posés sur ntdll.dll. Plutôt que d'appeler les fonctions de l'API Windows (qui sont hookées), le code malveillant appelle directement les instructions système kernel (syscalls) en hardcodant leurs numéros ou en les résolvant dynamiquement. La technique a été popularisée par les outils SysWhispers et Hell's Gate.

```
 ; Exemple de syscall direct NtAllocateVirtualMemory (x64 NASM)
; L'instruction syscall court-circuite ntdll.dll et ses hooks EDR
; Le numéro de syscall varie selon la version de Windows (doit être résolu dynamiquement)
NtAllocateVirtualMemory:
    mov r10, rcx          ; sauvegarde du premier argument
    mov eax, 0x18         ; numéro de syscall NtAllocateVirtualMemory (Win11 23H2)
    syscall               ; appel direct au kernel
    ret
```

Les **Indirect Syscalls** vont encore plus loin en exécutant l'instruction syscall depuis l'adresse correcte dans ntdll.dll (pour bypasser les protections basées sur la vérification de l'adresse de retour), tout en évitant les hooks posés sur les prologs de fonctions. Les outils offensifs modernes comme BruteRatel C4, Havoc C2, et certains loaders Cobalt Strike utilisent ces techniques.

Détection : Les EDR de 2026 utilisent plusieurs techniques pour détecter les syscalls directs : surveillance des appels système qui ne proviennent pas d'adresses dans ntdll.dll (Call Stack Analysis), heuristiques sur les patterns d'utilisation des numéros de syscall hors plage attendue, et corrélation avec d'autres signaux comportementaux.

Obfuscation et chiffrement des loaders : signature statique vs. détection dynamique

La bataille entre les loaders offensifs et la détection statique des EDR a conduit à l'adoption généralisée de techniques d'obfuscation avancées. En 2026, les loaders sophistiqués combinent plusieurs couches :

Chiffrement XOR ou AES du payload : le shellcode est chiffré en mémoire et déchiffré juste avant l'exécution, rendant la détection statique inutile

Payload Segmentation : découper le payload en petits chunks chiffrés séparément, assemblés au runtime

String Obfuscation : les chaînes de caractères révélatrices (noms de DLLs, fonctions API) sont obfusquées ou résolues dynamiquement via des hash personnalisés

Anti-debugging et Anti-sandbox : détection des environnements d'analyse (timing attacks, vérification de l'uptime, détection VM, vérification du nombre de cœurs CPU)

Code Signing : signature du loader avec un certificat de signature de code volé ou acheté pour passer les contrôles de confiance de niveau basique

Réponse défensive : La détection statique seule est insuffisante. Les EDR modernes combinent analyse statique (signatures, machine learning) et analyse dynamique (sandbox comportementale, emulation en mémoire). Des solutions comme Intezer Analyze ou Cape Sandbox permettent d'analyser les loaders suspects dans un environnement isolé pour identifier les techniques d'obfuscation.

Comparatif des principales solutions EDR/XDR face aux bypass 2026

Solution	Process Hollowing	BYOVD	ETW Patching	AMSI Bypass	Direct Syscalls
CrowdStrike Falcon Ultra	Détection élevée	HVCI requis + blocklist	Détecté (kernel)	Détecté	Détection partielle
Microsoft Defender XDR	Détection élevée	WDAC + HVCI	ETW-TI non patchable	Script Block Log	Heuristique
SentinelOne Singularity	Très haute	Anti-tamper kernel	Détecté	Détecté	Analyse comportementale
Elastic Security	Haute	Règles YARA kernel	Partiel	Via AMSI integration	Règles comportementales
Palo Alto Cortex XDR	Haute	BIOC rules	Détecté	Haute	ML comportemental

Ce tableau est indicatif et basé sur les tests publics disponibles (RedCanary, MITRE ATT&CK Evaluations). Les performances réelles dépendent fortement de la configuration, des règles personnalisées et de la version du produit. Les évaluations MITRE ATT&CK Enterprise constituent la référence la plus fiable pour comparer objectivement les capacités de détection.

Memory Scanning Evasion : contourner l'analyse de la mémoire en temps réel

Les EDR modernes effectuent des scans périodiques de la mémoire des processus pour détecter des patterns malveillants (shellcodes, PE injectés, IOC mémoire). Les attaquants ont développé plusieurs approches pour éviter cette détection :

Sleep Obfuscation : chiffrer le payload en mémoire pendant les périodes d'inactivité (entre les callbacks C2) et le déchiffrer juste avant l'exécution. Des techniques comme Ekko, Foliage ou Cronos utilisent des timers Windows ou des APC pour effectuer ce chiffrement/déchiffrement de manière asynchrone sans appeler VirtualProtect de manière suspecte.

Heap Spray puis Nettoyage : allouer le shellcode dans le heap, l'exécuter, puis nettoyer la mémoire immédiatement après — la fenêtre de détection est réduite à quelques millisecondes.

RWX vers RX Transition : les EDR alertent sur les régions mémoire avec permissions Read/Write/Execute (RWX) simultanées. Allouer en RW, copier le payload, puis passer en RX avant l'exécution pour éviter l'alerte RWX.

Contre-mesure : Les EDR avancés utilisent la technique dite de "memory scanning on allocation" — scanner la mémoire au moment de l'allocation (NtAllocateVirtualMemory hook) plutôt qu'en différé. L'utilisation de Microsoft Defender for Endpoint avec les capacités de memory forensics avancées (règles ASR) permet de détecter la plupart de ces techniques.

```
 # Règles ASR (Attack Surface Reduction) Microsoft Defender - à activer
# Block execution of potentially obfuscated scripts
Set-MpPreference -AttackSurfaceReductionRules_Ids "5BEB7EFE-FD9A-4556-801D-275E5FFC04CC" `
    -AttackSurfaceReductionRules_Actions Enabled

# Block process creations originating from PSExec and WMI commands
Set-MpPreference -AttackSurfaceReductionRules_Ids "D1E49AAC-8F56-4280-B9BA-993A6D77406C" `
    -AttackSurfaceReductionRules_Actions Enabled

# Block injection into lsass.exe
Set-MpPreference -AttackSurfaceReductionRules_Ids "9E6C4E1F-7D60-472F-BA1A-A39EF669E4B0" `
    -AttackSurfaceReductionRules_Actions Enabled
```