Análisis Técnico de AmsiScanBuffer: Mecanismos de Evasión, Defensas Avanzadas y el Horizonte de Amenazas 2026

1. Introducción Estratégica a la Arquitectura de AMSI

La Antimalware Scan Interface (AMSI) no debe considerarse un producto, sino una vulnerabilidad estructural por diseño. Al operar como el puente crítico entre los hosts de scripts (PowerShell, .NET, WMI) y los motores de detección, su posición en el User Mode garantiza una paridad de integridad entre el protector y el atacante. Esta arquitectura permite que el malware, al ejecutarse en el mismo espacio de memoria que la librería amsi.dll, pueda manipular el flujo de telemetría antes de que la carga útil sea siquiera evaluada.

El flujo operativo desde AmsiInitialize hasta AmsiScanBuffer es la cadena de custodia de la visibilidad en Windows. AMSI unifica la superficie de ataque al ser agnóstico respecto al proveedor; una técnica de evasión exitosa contra la interfaz neutraliza por igual a Microsoft Defender que a cualquier EDR de terceros. Esta integración abarca componentes vitales:

- PowerShell 5.1+ : Desofuscación dinámica de bloques de scripts.
- Framework .NET 4.8+ : Inspección de ensamblados cargados mediante reflexión.
- WMI (Windows Management Instrumentation): Análisis de persistencia en el repositorio.
- Exchange Server: Desde agosto de 2025, AMSI realiza el escaneo del cuerpo de solicitudes HTTP, aunque con una limitación crítica de 4KB (4096 bytes), un "punto ciego" que los atacantes explotan mediante el padding de payloads.

Para neutralizar esta defensa, el adversario no ataca el motor de detección, sino la función de transporte más crítica de la librería: AmsiScanBuffer.

2. Desconstrucción Técnica de AmsiScanBuffer y la Técnica de "Patching"

La manipulación de memoria en tiempo de ejecución busca alcanzar la "saturación de telemetría" o el silencio total. Dado que amsi.dll reside en el espacio de direcciones del proceso (como powershell.exe), el malware puede "silenciar al policía" mediante el parcheo de instrucciones. El acceso al mismo nivel de privilegios permite eludir la protección de solo lectura de las páginas de código mediante la API VirtualProtect.

Siguiendo el caso de estudio de CyberArk, el proceso de parcheo trasciende la simple devolución de un valor benigno. Se busca forzar un estado de error que el host ignore por defecto.

1. Localización: Uso de GetProcAddress para mapear la dirección de AmsiScanBuffer o AmsiScanString.
2. Modificación de Permisos: Cambio de permisos a PAGE_EXECUTE_READWRITE (0x40).
3. Patching en dos etapas: A diferencia de las evasiones genéricas, la vanguardia utiliza una lógica de error forzado.

INSTRUCCIÓN DE PARCHEO	OPCODE (HEX)	RESULTADO DE EVASIÓN
XOR RDX, RDX	48 31 d2	Etapa 1: Invalida el argumento de la cadena, forzando un fallo de integridad de argumentos.
MOV EAX, 0X0; RET	b8 00 00 00 00 c3	Etapa 2: Sobrescribe el código de error (0x80070057) con S_OK (0). El sistema cree que el escaneo fue exitoso y el búfer está limpio.

XOR EAX, EAX; RET	31 c0 c3	Forzado de retorno benigno directo sin llegar a llamar al motor de escaneo.

Este método engaña al sistema operativo haciéndole creer que la inspección ocurrió, cuando en realidad la lógica interna de amsi.dll ha sido cortocircuitada.

3. Metodologías de Evasión Avanzadas: Del "Patchless" al Smuggling

A medida que los EDRs han implementado monitoreo sobre VirtualProtect y NtProtectVirtualMemory, la ofensiva ha evolucionado hacia técnicas "patchless" que evitan la modificación de bytes ejecutables, eliminando los disparadores de telemetría comunes.

VEH² (Vectored Exception Handling) y Registros de Depuración: Esta técnica utiliza los registros de depuración de hardware del CPU (Dr0-Dr7) para establecer puntos de interrupción en AmsiScanBuffer. Al intentar ejecutar la función, se genera una excepción que es capturada por un manejador (VEH) controlado por el atacante. Este manejador redirige el flujo de ejecución hacia una instrucción ret benigna sin alterar un solo byte de la DLL en memoria. No obstante, los defensores estratégicos ahora monitorean NtSetContextThread mediante ETW Threat Intelligence (ETW-TI) para detectar la manipulación de estos registros.

ScriptBlock Smuggling: Representa una manipulación sofisticada del Árbol de Sintaxis Abstracta (AST) de PowerShell. El atacante construye un objeto ScriptBlock con una dualidad táctica:

- Atributo Extent: Contiene el código que AMSI recibe y escanea (ej. Write-Host "Safe").
- EndBlock: Contiene el código real que el compilador de PowerShell ejecuta (el payload malicioso).

El "So What?" estratégico: Esta técnica no solo evade la detección en tiempo real, sino que invalida los registros forenses. Los logs de Script Block Logging mostrarán el contenido del "Extent" benigno, ocultando la actividad maliciosa real incluso en un análisis post-mortem.

4. Ingeniería Social de Ejecución: El Fenómeno "ClickFix"

ClickFix es la respuesta táctica a la seguridad automatizada, delegando la evasión al usuario bajo el pretexto de una "solución técnica". Al involucrar al humano en el flujo de ejecución (human-in-the-loop), los atacantes evaden las protecciones que bloquean procesos iniciados por el navegador.

Cadena de Ataque y Lures Específicos: Los atacantes utilizan CAPTCHAs falsos o errores de "Cloudflare Turnstile" que emplean caracteres no latinos (UTF-8) para evadir detecciones basadas en expresiones regulares (regex). Los señuelos visuales presentan instrucciones para copiar un comando y pegarlo en el diálogo "Ejecutar" (Win+ R) o Windows Terminal.

Indicadores de Sospecha en Forensics: La clave de registro RunMRU es el campo de batalla forense. Se deben identificar patrones específicos:

- Uso de cadenas como: “I am not a robot - reCAPTCHA Verification ID: XXXX”.
- Ejecución de LOLBins como mshta.exe o powershell -eC (codificado) con caracteres de escape inusuales (^).
- Invocaciones directas de Invoke-RestMethod (irm) o Invoke-WebRequest (iwr) hacia dominios de corto ciclo de vida (.live, .shop, .icu).

La efectividad de ClickFix ha forzado su expansión hacia macOS, donde se utiliza para recolectar contraseñas mediante sudo -S tras engañar al usuario con falsas actualizaciones de sistema. Esta vulnerabilidad humana hace que la protección exclusiva en el endpoint sea una estrategia insuficiente.

5. Arquitectura Defensiva y Mitigación Crítica

Ante un adversario que manipula la memoria y el consentimiento del usuario, la defensa debe ser arquitectónica, no basada en firmas. La mitigación nativa definitiva es el Constrained Language Mode (CLM).

El "Checkmate" de CLM: CLM es la medida más potente porque bloquea directamente los métodos .NET (Reflexión y P/Invoke) necesarios para llamar a VirtualProtect o GetProcAddress. Sin acceso a estas APIs, las técnicas de parcheo analizadas en las secciones 2 y 3 quedan neutralizadas. La aplicación de CLM debe realizarse mediante WDAC (Windows Defender Application Control), que garantiza que solo scripts firmados digitalmente operen con privilegios de lenguaje completo.

Checklist Estratégico de Recomendaciones:

1. Visibilidad: Habilitar Script Block Logging y Module Logging para capturar la desofuscación final.
2. Telemetría de Bajo Nivel: Implementar soluciones que utilicen ETW-TI para monitorear llamadas a NtProtectVirtualMemory y manipulaciones de contexto de hilo (NtSetContextThread).
3. Reducción de Superficie (ASR): Bloquear la ejecución de scripts ofuscados y, si no es estrictamente necesario, deshabilitar el diálogo "Ejecutar" (Win+ R) vía GPO.
4. Integridad de Proveedores: Exigir que todos los proveedores de AMSI estén firmados mediante Authenticode (requerido desde Windows 10 1903).

6. Horizonte 2026: La "IA-ficación" de las Amenazas y el Fin del Modelo Tradicional

Para el año 2026, el modelo de seguridad basado en firmas de AMSI será obsoleto. Entraremos en la era del polimorfismo a escala impulsado por IA, donde cada cadena de comando será única y generada en milisegundos.

Amenazas Emergentes:

- Vibe Coding y Backdoors de IA: La generación acelerada de código introducirá vulnerabilidades inadvertidas en procesos críticos, creando puntos de entrada para la IA Agéntica.
- Ransomware Autónomo: Agentes de IA realizarán reconocimientos y negociaciones de forma independiente, manipulando flujos de confianza sin intervención humana.

Cambios Estratégicos (Strategic Shifts):

- De Firmas Estáticas a Verificación de Intención: AMSI ya no buscará "patrones", sino que deberá integrarse con modelos de IA defensivos que evalúen la "intención" del flujo de ejecución.
- De Defensa de Endpoint a Gobernanza de Identidad: La resiliencia dependerá de controlar el consentimiento y las autorizaciones (OAuth worms, pivoteo SaaS-to-SaaS) más que el código ejecutado.
- Gobernanza de Consentimiento: Ante tácticas como ClickFix, la identidad y la verificación de la procedencia de la acción serán los únicos pilares de confianza restantes.

Análisis Avanzado de Evasión de Seguridad en Windows: Ataque a AMSI, Ofuscación y Manipulación de Memoria

En mi experiencia analizando arquitecturas de seguridad, la Antimalware Scan Interface (AMSI) es el último bastión de visibilidad en el espacio de usuario. No es una ley de seguridad, sino una sugerencia técnica: un puente que conecta los hosts de scripting con los motores de detección para inspeccionar payloads *fileless* antes de su ejecución. Sin embargo, AMSI presenta una paradoja arquitectónica fundamental: al residir en el *user-mode*, el "policía" (la DLL) comparte el mismo espacio de memoria y privilegios que el "sospechoso" (el proceso malicioso). Para un Arquitecto de Red Team, esto significa que la integridad de AMSI es, por diseño, un estado de confianza que podemos —y debemos— revocar.

1. Fundamentos Arquitectónicos de AMSI (Antimalware Scan Interface)

AMSI opera bajo un modelo proveedor-consumidor agnóstico al motor de antivirus. El sistema confía en que el proceso consumidor informe fielmente sobre su propia actividad.

- Consumidores: Componentes como PowerShell 5.1+ , Office VBA (Macros), .NET 4.8+ y, recientemente, Exchange Server (actualización de agosto 2025 para escanear cuerpos de solicitudes HTTP de hasta 1MB).
- Interfaz (amsi.dll): La biblioteca cargada en el proceso que facilita el escaneo. Si forzamos su fallo en la inicialización, el sistema a menudo cae en un estado de "Default Allow" por razones de disponibilidad.
- Proveedores: Motores como Windows Defender que implementan IAntimalwareProvider para analizar los buffers recibidos.

FUNCIÓN API	ROL OPERATIVO	IMPORTANCIA ESTRATÉGICA PARA EVASIÓN
AMSIINITIALIZE	Inicia el contexto de AMSI.	Si el handle devuelto es nulo, muchas aplicaciones omiten el escaneo por completo.
AMSISCANBUFFER	Escanea buffers de datos en bruto.	Objetivo principal para el parcheo de registros y redirección del flujo en memoria.
AMSISCANSTRING	Analiza scripts basados en texto.	Vulnerable a manipulación de registros de argumentos para forzar errores de integridad.
AMSIRESULTISMALWARE	Evalúa si el resultado es malicioso.	Modificar esta lógica permite que un resultado AMSI_RESULT_DETECTED se interprete como benigno.

Capa del "So What?": La debilidad estratégica de AMSI radica en su dependencia de la autoevaluación. Un proceso comprometido tiene la autoridad técnica para desarmar los ganchos (hooks) de AMSI antes de que el primer byte malicioso sea analizado. En el momento en que tomamos el control del espacio de memoria, el "puente de visibilidad" se convierte en un punto ciego.

2. El Arte de la Ofuscación: Superando la Detección por Firmas

La evolución de las firmas ha obligado a los operadores de Red Team a tratar el código no como lógica, sino como datos maleables. Los motores modernos ya no solo buscan hashes, sino patrones específicos y comportamientos.

- Manipulación de Cadenas: La concatenación ("Inv" + "oke") y el uso de *backticks* (`) en PowerShell son tácticas básicas. Sin embargo, el reordenamiento con operadores de formato ("{1}{0}" -f "katz", "mimi") sigue siendo efectivo contra firmas estáticas simples.
- Codificación Dinámica: El uso de Base64, Hexadecimal o codificaciones personalizadas oculta el payload, pero genera una huella de "Alta Entropía".
- Automatización con Invoke-Obfuscation: Herramientas que randomizan nombres de variables y estructuras de control para generar cadenas únicas por cada ejecución.

Capa del "So What?": La cruda realidad es que la ofuscación excesiva es en sí misma una firma. Los EDR modernos utilizan heurísticas para detectar "Indicadores de Ofuscación". Un script con nombres de variables de un solo carácter y alta densidad de caracteres especiales activará alarmas de comportamiento, independientemente de si la firma del malware es reconocida. La verdadera maestría consiste en la ofuscación que parece actividad administrativa legítima.

3. Manipulación de Memoria y Parcheo de amsi.dll

Cuando la ofuscación de texto falla, la intervención quirúrgica en la memoria de la librería amsi.dll es el siguiente paso lógico. El objetivo es tomar el control de las funciones de escaneo una vez que la DLL ha sido cargada.

Precisión de Ensamblador: El Parcheo de AmsiScanString

Siguiendo las investigaciones de CyberArk, no basta con un parche genérico. Un ataque de alta precisión sobre AmsiScanString implica:

1. Localización: Identificar la dirección mediante GetProcAddress.
2. Ajuste de Permisos: Cambiar la memoria a PAGE_EXECUTE_READWRITE con VirtualProtect.
3. Manipulación de Registros:
    a. Zeroing del registro rdx (instrucción xor rdx, rdx / opcodes 48 31 d2). Esto provoca que la función detecte un argumento inválido y salte a la lógica de error.
    b. Modificación del registro eax a 0. Originalmente, el error devolvería E_INVALIDARG (0x80070057), pero al forzarlo a 0 (S_OK), el sistema asume que el escaneo fue exitoso y el contenido es benigno.

Reflexión de .NET y amsiInitFailed

Históricamente, la técnica de Matt Graeber de modificar el campo amsiInitFailed en la clase System.Management.Automation.AmsiUtils mediante reflexión era la norma. Al establecer este campo como $true, se engaña al host para que crea que AMSI nunca se inició correctamente, desactivando los escaneos posteriores.

Capa del "So What?": Estos parches son extremadamente efectivos pero "ruidosos". Los EDR de última generación monitorean específicamente las llamadas a VirtualProtect sobre bibliotecas del sistema y verifican la integridad de los bytes en memoria de amsi.dll. Modificar los bytes ejecutables es hoy una sentencia de muerte para el sigilo si no se combina con técnicas de desenganche (unhooking).

4. Evasión "Patchless" y Técnicas Dinámicas Avanzadas

Para eludir el monitoreo de integridad de memoria, debemos evitar la alteración binaria. Aquí es donde entramos en el terreno de la evasión *patchless*.

- VEH² y Hardware Breakpoints: En lugar de cambiar los bytes de AmsiScanBuffer, utilizamos los registros de depuración del CPU (Dr0–Dr7) para establecer un *breakpoint* de hardware. Cuando el hilo intenta ejecutar la función de escaneo, se dispara una excepción que es capturada por un *Vectored Exception Handler* (VEH) registrado por nosotros, el cual redirige el flujo sin modificar la DLL.
- CLR Hosting y ProvideAssembly: Al cargar el *Common Language Runtime* manualmente mediante interfaces como IHostAssemblyStore, podemos inyectar ensamblados de C# a través de la ruta ProvideAssembly. Esta vía documentada evita los disparadores internos de clr.dll que normalmente activan a AMSI durante la carga de ensamblados.
- ScriptBlock Smuggling (Manipulación de AST): Esta técnica explota la dualidad del objeto *ScriptBlock* en PowerShell. Manipulamos el Árbol de Sintaxis Abstracta para que el atributo Extent (lo que AMSI escanea y los logs registran) contenga código benigno, mientras que el EndBlock compilado (lo que realmente se ejecuta) contiene el payload malicioso.

Capa del "So What?": Estas técnicas explotan funciones legítimas para ocultar la intención. Mientras que NtSetContextThread para configurar registros DrX es monitoreado por ETW-TI, los atacantes avanzados están utilizando NtContinue para actualizar el contexto del hilo de forma más sigilosa, evitando los puntos de control tradicionales de inteligencia de amenazas del kernel.

5. Ingeniería Social y el Método "ClickFix"

El auge de ClickFix traslada la carga de la ejecución del exploit automatizado a la acción consciente del usuario, abusando de interfaces de confianza.

- Ciclo del Ataque: Un señuelo (falso CAPTCHA o error de Office) instruye al usuario a presionar Win + R y pegar un comando "reparador".
- Abuso de la Interfaz "Ejecutar": Los atacantes prefieren el diálogo Run porque carece de las advertencias de seguridad multilínea que la Terminal de Windows ha implementado recientemente. Además, el comando se ejecuta bajo el contexto del usuario con confianza implícita.
- Expansión a macOS: Se ha observado el uso de ClickFix para entregar el malware AMOS. El flujo implica curl -o /tmp/update, otorgar permisos con chmod +x, y engañar al usuario para que valide su password mediante dscl . -authonly y luego ejecutar con sudo -S xattr -c para eludir las protecciones de cuarentena de macOS.

Capa del "So What?": En estos escenarios, los controles técnicos son secundarios. La identidad ha sido secuestrada para realizar acciones "administrativas" legítimas. Para el investigador forense, el rastro definitivo se encuentra en la clave de registro HKEY_CURRENT_USER₩Software₩Microsoft₩Windows₩CurrentVersion₩Explorer₩RunMRU, que almacena la historia de comandos ejecutados en el diálogo Run, convirtiéndose en la fuente de verdad sobre el compromiso inicial.

6. Arquitecturas de Defensa y Hardening del Entorno

Un Red Team Architect debe conocer la defensa para superarla. La seguridad no es un estado, sino un proceso de reducción de superficie.

- PowerShell Constrained Language Mode (CLM): Es la defensa más robusta. Al activarse (normalmente vía AppLocker/WDAC), bloquea funciones críticas para la evasión, incluyendo:
  - Namespaces: System.Reflection, System.Runtime.InteropServices (bloquea P/Invoke).
  - Métodos: System.Convert (evita decodificación Base64 simple) y creación de objetos COM arbitrarios.
  - Comando de activación sugerido: [Environment]::SetEnvironmentVariable('__PSLockdownPolicy', '4', 'Machine').
- Implementación de WDAC: Forzar políticas donde solo scripts firmados por certificados de confianza puedan ejecutarse en *Full Language Mode*.
- ETW-TI (Event Tracing for Windows - Threat Intelligence): Monitoreo desde el kernel de llamadas sospechosas como NtSetContextThread o el uso inusual de NtContinue.

Capa del "So What?": La seguridad efectiva requiere visibilidad profunda (logs de ScriptBlock) y restricciones de ejecución estrictas. La combinación de CLM y WDAC elimina el 90% de los vectores de ataque AMSI-centric al despojar al atacante de las herramientas de introspección necesarias para manipular la memoria.

7. Hacia el 2026: La "AI-ficación" de las Amenazas

Para 2026, la capacidad de detectar "firmas" será irrelevante. Entraremos en la era de la "Intención" y la automatización autónoma.

- Polimorfismo Impulsado por IA y Ataques "Vibe-Coded": Los atacantes generarán cadenas de comandos únicas por segundo. Cada instancia del ataque tendrá una estructura distinta, invalidando las bases de datos de firmas de AMSI en tiempo real.
- Agentes de IA Autónomos: Entidades que realizarán reconocimiento y movimiento lateral a velocidades inhumanas, negociando rescates de forma autónoma basándose en la sensibilidad de los datos exfiltrados.
- OAuth Worms: El compromiso se desplazará del endpoint a la gobernanza de identidades SaaS. Los gusanos de consentimiento abusarán de permisos de aplicaciones en la nube, evadiendo por completo a AMSI al no requerir ejecución local de scripts tradicionales.

Conclusión: En 2026, la única capa de verdad viable será el análisis de comportamiento y la gobernanza estricta de identidades. Ante payloads infinitamente variables generados por IA, el enfoque debe pasar de inspeccionar el *qué* (el código) a monitorear el *quién* (la identidad) y el *por qué* (el comportamiento). En este nuevo ecosistema, la visibilidad ya no es un lujo, sino el único mecanismo de supervivencia.

Guía de Metodología Ofensiva: Evasión Avanzada de AMSI y Red Teaming en Memoria

1. Fundamentos Arquitectónicos: El Rol de AMSI en el Ecosistema Moderno

El Antimalware Scan Interface (AMSI) representa el puente crítico de visibilidad en el tiempo de ejecución de Windows. No es un motor de detección *per se*, sino una interfaz estandarizada que permite a los hosts de scripting (PowerShell, WSH, .NET 4.8+ , VBA) enviar contenido dinámico deofuscado a los proveedores de seguridad registrados (como Windows Defender o EDRs de terceros) antes de su ejecución. Su importancia estratégica radica en su capacidad para interceptar cargas útiles "fileless" en el preciso instante en que la memoria del proceso las reconstruye en texto plano, neutralizando la efectividad de la ofuscación en disco.

Arquitectónicamente, AMSI opera bajo un modelo de proveedor-consumidor. El flujo operativo se inicia con AmsiInitialize, que carga la infraestructura de escaneo, seguido de AmsiOpenSession. Los buffers de datos se remiten a AmsiScanBuffer o AmsiScanString, donde el proveedor realiza el análisis. Es imperativo comprender que AMSI solo emite un veredicto; la ejecución de la política recae en la aplicación host (ej. powershell.exe), que decide si bloquear el hilo basándose en si el resultado alcanza el umbral AMSI_RESULT_DETECTED.

La paradoja táctica de AMSI es su residencia íntegra en el espacio de usuario (user-mode). Al cargarse amsi.dll en el mismo espacio de memoria virtual y nivel de integridad que el proceso sospechoso, el sistema carece de aislamiento real. Esta arquitectura permite que cualquier código con capacidad de ejecución en el proceso pueda "silenciar al policía" mediante la manipulación directa de la librería, explotando la vulnerabilidad inherente de un monitor que comparte privilegios con el monitoreado.

Funciones Principales de la API AMSI

FUNCIÓN API	ROL OPERACIONAL	RELEVANCIA TÁCTICA EN LA EVASIÓN
AMSIINITIALIZE	Inicializa el contexto y consulta proveedores en el registro.	Punto de interrupción para abortar la carga del entorno de escaneo.
AMSIOPENSESSION	Establece el vínculo para flujos de datos correlacionados.	Forzar fallos aquí induce estados de "permitir por defecto".
AMSISCANBUFFER	Analiza buffers de memoria crudos.	Objetivo prioritario para parcheo y redirección de flujo.
AMSISCANSTRING	Analiza cadenas de texto de scripts.	Vulnerable a manipulación de argumentos de longitud y codificación.
AMSIRESULTISMALWARE	Evalúa si el veredicto indica una amenaza.	La alteración de esta lógica permite reportar malware como benigno.

La comprensión de esta jerarquía es el primer paso antes de agotar las vías de menor fricción: la evasión de firmas estáticas.

2. Ofuscación de Scripts y Evasión de Firmas Estáticas

La detección basada en firmas ha trascendido los hashes de archivos para centrarse en heurísticas de entropía e "Indicadores de Ofuscación". El uso de frameworks automatizados como Invoke-Obfuscation es contraproducente en operaciones de alto nivel, ya que generan patrones de alta entropía que disparan alertas *per se*. La ofuscación manual sigue siendo superior al permitir una manipulación quirúrgica de los triggers basados en regex que utiliza AMSI.

Operacionalmente, AMSI no solo busca strings estáticos como Invoke-Mimikatz; busca patrones de lógica de concatenación (ej. Set-Variable + OFS). Por tanto, el operador debe evitar patrones predecibles. La concatenación manual, el uso de backticks y el reordenamiento dinámico son herramientas tácticas para fragmentar firmas conocidas sin elevar la entropía del script a niveles sospechosos.

Evolución Táctica de un Payload (Ejemplo: Invoke-Mimikatz)

1. Forma Plana (Detección Inmediata): Invoke-Mimikatz
2. Concatenación Reactiva: $a = "Inv"; $b = "oke-Mi"; $c = "mikatz"; &($a+$b+$c)
3. Uso de Caracteres de Escape (Backticks): Invoke-Mimikatz
4. Codificación Base64 en Tiempo de Ejecución: $s = [System.Text.Encoding]::Unicode.GetString([System.Convert]::FromBase64String('SQBuAHYAbwBrAGUALQBNAGkAbQBpAGsAYQB0AHoA'))
5. Reordenamiento mediante Operador de Formato (-f): $m = "{1}{0}" -f 'katz','mimi'; &("Invoke-$m")

Para firmas críticas como amsiInitFailed (utilizada para forzar fallos de inicialización), la fragmentación es esencial para evitar el trigger inicial de carga. Si la ofuscación de strings es insuficiente frente a motores con análisis profundo de ScriptBlocks, la metodología dicta el descenso hacia la manipulación de la librería en memoria.

3. Parcheo de Memoria: Neutralización de AmsiScanBuffer

El parcheo en caliente (*hot-patching*) de amsi.dll es una técnica de neutralización definitiva que fuerza estados de confianza artificiales. Dado que todo flujo de scripting debe transitar por AmsiScanBuffer, la sobrescritura de sus instrucciones iniciales permite abortar el escaneo antes de que el buffer sea procesado por el proveedor de seguridad.

Metodología de Parcheo Técnico (5 Pasos)

1. Localización: Identificar la dirección base de amsi.dll y el offset de AmsiScanBuffer mediante GetProcAddress.
2. Ajuste de Permisos: Modificar la protección de la memoria de la sección .text de PAGE_EXECUTE_READ a PAGE_EXECUTE_READWRITE (0x40) usando VirtualProtect.
3. Sobrescritura: Inyectar opcodes de ensamblador que manipulen el registro de retorno (EAX/RAX).
4. Restauración: Reinstaurar los permisos originales para evadir escaneos básicos de integridad de memoria.
5. Operatividad: Ejecutar el payload malicioso en una sesión ahora "ciega".

Comparativa de Estrategias de Parcheo

ESTRATEGIA	CÓDIGO HEXADECIMAL	IMPACTO EN REGISTRO	"SO WHAT?": JUSTIFICACIÓN TÁCTICA
RETORNO S_OK	31 c0 c3	xor eax, eax; ret	Reporta un escaneo exitoso sin hallazgos. Vulnerable a unhooking simple.
INYECCIÓN DE ERROR	b8 57 00 07 80 c3	mov eax, 0x80070057; ret	Reporta E_INVALIDARG. Induce un "permitir por defecto" por fallo estructural del host.

La inyección de E_INVALIDARG ofrece una resiliencia superior frente a EDRs modernos, ya que simula un error de paso de argumentos que el consumidor suele ignorar para mantener la estabilidad de la aplicación, a diferencia de un S_OK artificial que puede ser contrastado mediante telemetría de integridad de memoria.

4. Evasión "Patchless": Hardware Breakpoints y VEH[2]

La evolución de los sensores de integridad de memoria en EDRs de próxima generación obliga al abandono de la modificación de bytes ejecutables. La evasión "patchless" utiliza la arquitectura de depuración del CPU (registros Dr0-Dr7) y el *Vectored Exception Handling* (VEH) para redirigir el flujo sin alterar la integridad de la DLL en disco o memoria.

Flujo Lógico de Intercepción VEH[2]

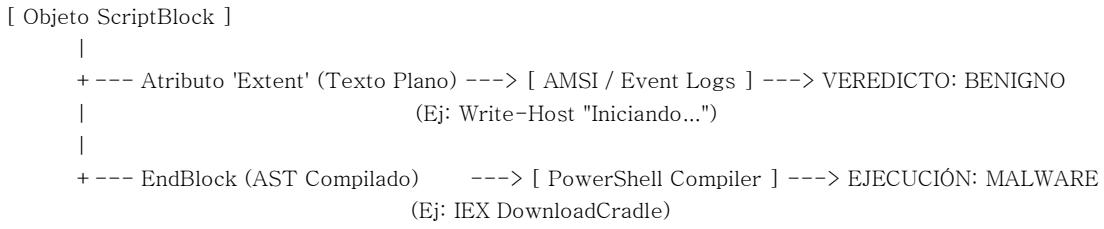
- Registro de Handlers: Se establece un handler de excepciones prioritario en el proceso.
- Configuración de Breakpoint: Se definen los registros de depuración para disparar una excepción al alcanzar la dirección de AmsiScanBuffer.
- Intercepción de Excepción: El CPU genera una excepción de depuración ("Single Step") antes de ejecutar la instrucción.
- Modificación de Contexto: El VEH intercepta la excepción y modifica el registro de instrucción (RIP) y el de retorno (RAX) para saltar la lógica de escaneo.

Riesgo Operativo y Mitigación: El uso de SetThreadContext es una señal de alta fidelidad monitoreada por ETW Threat Intelligence (ETW-TI) a través de NtSetContextThread. Para mitigar esta telemetría, el operador debe emplear NtContinue para actualizar el contexto del hilo de forma más encubierta, evitando los ganchos tradicionales de monitoreo de hilos que emplean la mayoría de los EDRs.

5. ScriptBlock Smuggling y Manipulación de AST

El ataque estratégico hacia el *Abstract Syntax Tree* (AST) busca crear una discrepancia operativa entre lo que el sistema audita y lo que el procesador ejecuta. El ScriptBlock Smuggling explota la dualidad de los objetos ScriptBlock en PowerShell, separando su representación de escaneo de su lógica de ejecución.

Diagrama de Discrepancia AST (Escaneo vs. Ejecución)



Tácticamente, esta técnica es devastadora para el análisis forense post-incidente. Al manipular el AST, los registros de Event Log 4104 (ScriptBlock Logging) mostrarán comandos inofensivos, mientras que el motor de ejecución procesa el payload malicioso, falsificando la línea de tiempo forense.

6. CLR Hosting y Evasión en la Carga de Ensamblados

Desde .NET 4.8, el escaneo automático de ensamblados en memoria mediante Assembly.Load es la norma. Para evadir la construcción del objeto RawImageLayout que dispara AMSI, es táctico hostear el *Common Language Runtime* (CLR) desde un binario C++ personalizado.

Mediante la implementación de las interfaces ICLRRuntimeHost e IHostAssemblyStore, el operador asume el control del suministro de datos al runtime.

Metodología de Suministro Manual (ProvideAssembly)

1. Instanciación del Host: Iniciar el CLR dentro de un proceso legítimo de alta confianza.
2. Intercepción de Carga: Utilizar IHostAssemblyStore para interceptar las solicitudes de carga de ensamblados.
3. Desencriptación JIT: Mantener los payloads .NET encriptados en memoria (at-rest) hasta el último nanosegundo.
4. Inyección Directa: Proveer el buffer desencriptado mediante el método ProvideAssembly, evitando las rutas de carga convencionales monitoreadas por el proveedor de AMSI.

Esta arquitectura ofrece una ventaja competitiva crítica: los programas .NET poseen representaciones en memoria que difieren significativamente de sus versiones en disco, lo que hace que el análisis de firmas por parte de un AV sea inherentemente difícil una vez que el ensamblado ha sido inyectado a través de rutas no instrumentadas.

7. Defensa de Plataforma y Evaluación de Resiliencia en Windows 11

Windows 11 (23H2/24H2 y el futuro 25H2) ha endurecido la plataforma para reducir la superficie de ataques *Living-off-the-Land* (LotL). La implementación de Smart App Control (SAC) bloquea por defecto cargadores no firmados o de baja reputación, actuando como un filtro preventivo antes de que AMSI sea invocado. Además, la versión 25H2 marca la eliminación definitiva de WMIC y componentes legados como PowerShell 2.0 (que carecía de AMSI), eliminando vectores de ataques de "downgrade".

A pesar de estas mejoras, la mitigación más robusta contra los bypasses discutidos es el Constrained Language Mode (CLM) forzado mediante WDAC. CLM neutraliza el uso de reflexión, tipos .NET arbitrarios y acceso a la API Win32, invalidando la base técnica de casi todos los bypasses de memoria.

Matriz de Evasión vs. Detección (Escenario EDR + ETW-TI)

TÉCNICA DE EVASIÓN	DIFICULTAD DE DETECCIÓN	VECTOR DE IDENTIFICACIÓN
OFUSCACIÓN ESTÁTICA	Baja / Media	Heurísticas de Entropía y ScriptBlock Logging.
MEMORY PATCHING	Media / Alta	Scans de integridad de DLLs y hooks en VirtualProtect.
VEH / NTCONTINUE	Alta	Monitoreo de NtSetContextThread via ETW-TI.
MANIPULACIÓN AST	Crítica	Análisis conductual y discrepancia de logs forenses.

Recomendaciones Estratégicas para Defensores

- WDAC es Mandatorio: El cumplimiento de CLM es la única defensa efectiva contra técnicas basadas en reflexión y parcheo de memoria.
- Logging de ScriptBlocks: Activar el registro profundo (ID 4104) para capturar la lógica deofuscada, independientemente del estado de AMSI.
- Monitoreo de Integridad de DLLs: Implementar verificaciones dinámicas que comparen la copia en memoria de amsi.dll y ntdll.dll con su versión en disco.
- Visibilidad ETW-TI: Emplear soluciones que instrumenten eventos a nivel de kernel para identificar la manipulación de contextos de hilos y el uso de registros de depuración del hardware.

Marco Estratégico de Seguridad: Endurecimiento de Infraestructura Windows contra la Evasión de AMSI (2025-2026)

1. Contexto Estratégico: El Nuevo Paradigma de Amenazas 2025-2026

En el actual panorama de ciberseguridad, la Antimalware Scan Interface (AMSI) ha dejado de ser un componente complementario para consolidarse como la última línea de defensa crítica. En un ecosistema dominado por la ejecución íntegramente en memoria y la "AI-fication" de las amenazas, AMSI actúa como el sensor arquitectónico fundamental entre el contenido dinámico y los motores de detección. Las firmas estáticas son hoy irrelevantes ante el auge del "Vibe Coding", donde agentes autónomos generan cadenas de comandos polimórficas únicas en milisegundos, adaptando el payload al contexto específico del objetivo antes de que cualquier sistema de reputación pueda catalogarlo.

Esta velocidad técnica se combina con la sofisticación de la ingeniería social mediante la técnica "ClickFix". Esta táctica induce al usuario a resolver supuestos errores técnicos ejecutando comandos directamente en el diálogo "Ejecutar" (Win+ R) o la Terminal. Al forzar la ejecución en el espacio de proceso de explorer.exe, los atacantes utilizan a ClickFix como el vector de entrega primario para cargas "Vibe-coded", eludiendo las soluciones de seguridad perimetral y de correo que no pueden inspeccionar acciones directas del usuario. Además, la aparición de "AI Predator Swarms" introduce un riesgo de denegación de servicio en las capas de seguridad, utilizando variaciones masivas para agotar los recursos de escaneo o explotar los límites de inspección de los buffers de AMSI.

Imperativo Estratégico: La confianza ya no puede residir en el contenido, sino en la identidad y la procedencia. Ante ataques autónomos que superan la capacidad de respuesta humana, el endurecimiento de la infraestructura es la única respuesta viable. La defensa debe operar a nivel de arquitectura, asumiendo que el contenido será inherentemente malicioso y dinámico.

---

2. Anatomía de la Evasión: Del Parcheo de Memoria a las Técnicas "Patchless"

AMSI presenta una paradoja arquitectónica: es un mecanismo de seguridad de modo usuario vigilado por el mismo proceso que intenta monitorear. Dado que amsi.dll se carga en el espacio de memoria del proceso (PowerShell, macros de Office), cualquier código con el mismo nivel de integridad puede silenciar al "policía".

Evolución de Metodologías de Evasión

METODOLOGÍA	DESCRIPCIÓN TÉCNICA	OBJETIVO ARQUITECTÓNICO
OFUSCACIÓN Y REFLEXIÓN	Uso de .NET Reflection para manipular campos internos como amsiInitFailed.	Forzar un estado de fallo de inicialización para omitir escaneos.
PARCHEO DE MEMORIA	Sobrescritura de instrucciones en AmsiScanBuffer (ej. xor eax, eax; ret).	Retornar un código de éxito (S_OK) sin realizar el escaneo real.
TÉCNICAS PATCHLESS (VEH²)	Uso de registros de depuración de hardware (Dr0-Dr7) y *Vectored Exception Handling*.	Redirigir el flujo de ejecución sin modificar un solo byte de la DLL, evadiendo controles de integridad.

Vectored Exception Handling (VEH²) y Hardware Breakpoints

Las técnicas de vanguardia han abandonado el parcheo de bytes —fácilmente detectable por EDRs modernos— en favor de la manipulación del contexto del hilo. Mediante el uso de los registros de depuración de hardware Dr0 a

Dr7, los atacantes establecen puntos de interrupción en la dirección de AmsiScanBuffer. Al dispararse la excepción, un manejador (VEH) intercepta el flujo, modifica el registro de retorno y redirige la ejecución. Esta técnica es invisible para los escaneos de integridad de memoria tradicionales.

ScriptBlock Smuggling y Manipulación de AST

El ScriptBlock Smuggling explota la dualidad de las representaciones en PowerShell. Al manipular el Árbol de Sintaxis Abstracta (AST), los atacantes logran que el atributo *Extent* (lo que ven los registros y AMSI) muestre un comando benigno, mientras que el *EndBlock* (lo que realmente ejecuta el motor) contiene la carga maliciosa. Esta técnica representa un riesgo crítico para la forense digital, ya que falsifica el rastro de evidencias en los registros del sistema, permitiendo una persistencia indetectable post-incidente.

---

3. Pilar I: Implementación de Constrained Language Mode (CLM)

Architecture debe imponer el Modo de Lenguaje Restringido (CLM) como la barrera nativa más poderosa contra la reflexión y el parcheo de memoria. Al reducir drásticamente la superficie de ataque, CLM neutraliza la capacidad de un script para interactuar con APIs de bajo nivel.

Restricciones Operativas y su Impacto en la Evasión

Elemento Bloqueado	Impacto Directo en la Defensa
Llamadas a Win32 API	Impide el uso de VirtualProtect, imposibilitando el parcheo de AmsiScanBuffer.
Método .FromBase64String	Bloquea la decodificación nativa de .NET, forzando a los atacantes a usar técnicas de escape de caracteres menos eficientes.
Reflexión (.NET)	Elimina el acceso a AmsiUtils, neutralizando bypasses basados en amsiInitFailed.

Estrategia de Despliegue mediante AppLocker/WDAC

Es no-negociable que el CLM se gestione a través de políticas de Windows Defender Application Control (WDAC) o AppLocker basadas en certificados y editores, en lugar de configuraciones globales (__PSLockdownPolicy__) que degradan la operatividad administrativa. Architecture debe monitorear la creación de archivos temporales PSScriptPolicyTest*.ps1, utilizados por AppLocker para validar el modo de lenguaje; cualquier manipulación de estos archivos es un indicador de compromiso de alta prioridad. El uso de certificados de firma de código permite que los scripts corporativos operen en *Full Language Mode*, manteniendo la productividad mientras se protege el resto del ecosistema.

---

4. Pilar II: Monitoreo Avanzado mediante ETW Threat Intelligence (ETW-TI)

Para resolver la ceguera intrínseca del modo usuario, es obligatorio el despliegue de telemetría desde el kernel mediante el proveedor ETW Threat Intelligence (ETW-TI). Esta es la única vía para detectar las operaciones críticas que preceden a un bypass.

Detección de Vectores de Kernel Críticos

Architecture debe centrar la detección en tres vectores de alta fidelidad:

1. Modificaciones de Protección de Memoria: Monitoreo de NtProtectVirtualMemory cuando el objetivo es el rango de direcciones de amsi.dll.
2. Manipulación de Contexto de Hilo: Identificación de llamadas a NtSetContextThread o NtContinue dirigidas a los registros Dr0-Dr7. Estos son indicadores unívocos de un intento de bypass "patchless" (VEH²).
3. Tamper de Procesos: Detección de cargas reflectivas de ensamblados .NET y *process hollowing*.

La correlación de estos eventos de kernel con alertas de AMSI de modo usuario es el único método para lograr una detección de alta confianza. Sin esta visibilidad profunda, el adversario puede cegar al motor de seguridad y operar sin oposición.

---

5. Pilar III: Configuración Avanzada de Microsoft Exchange Server

La integración de AMSI en Exchange Server (actualizaciones de agosto 2025) debe tratarse como un Firewall de Aplicaciones (WAF) interno. Sin embargo, Architecture debe reconocer una limitación crítica: esta integración excluye el tráfico RPC sobre HTTP, dejando un vector de exposición que debe mitigarse con controles complementarios.

Parámetros de Configuración Obligatorios

Para una postura de seguridad robusta, se deben aplicar los siguientes ajustes:

- Ajuste del tamaño de escaneo: Incrementar el límite inicial de 4 KB hasta 1 MB para capturar payloads polimórficos complejos de "Predator Swarms".
- Comportamiento de Fallback: En entornos de alta seguridad, es imperativo cambiar el comportamiento por defecto de "Allow" a "Block". Si el motor antimalware no está disponible, el tráfico debe interrumpirse.
- HttpRequestFilteringModule: Auditar exhaustivamente los logs en %ExchangeInstallPath%₩Logging₩HttpRequestFiltering para detectar intentos de despliegue de *web shells*.

Esta configuración neutraliza la entrega de payloads web en el punto de entrada, mitigando vulnerabilidades de día cero y limitando el movimiento lateral.

---

6. Hardening de Plataforma: Windows 11 (24H2/25H2)

Las nuevas versiones de Windows 11 habilitan un entorno "Secure-by-Design", pero su implementación requiere precisión estratégica.

- Smart App Control (SAC): Es fundamental entender que SAC solo está disponible en instalaciones limpias (clean installs) de Windows 11. No puede habilitarse tras una actualización desde versiones anteriores, lo que debe dictar la estrategia de refresco de hardware y despliegue de imágenes.

- Eliminación de Componentes Legados: La arquitectura debe imponer la eliminación total de PowerShell 2.0 (vulnerable a ataques de downgrade) y WMIC. Estas medidas obligan a los atacantes a abandonar tácticas *Living-off-the-Land* clásicas y enfrentarse a capas de defensa modernas y altamente monitoreadas.

---

7. Conclusión: Hacia una Defensa Basada en la Intención

El endurecimiento contra la evasión de AMSI no es una simple tarea de parcheo, sino una lucha estratégica por el control del entorno de ejecución. La resiliencia en 2026 dependerá de una defensa basada en la intención, definida como la correlación sistemática de la telemetría de modo kernel (ETW-TI) con los registros de comportamiento de scripts de modo usuario.

Pasos Inmediatos para la Implementación:

1. Imponer CLM mediante WDAC: Establecer una política de "Denegar por Defecto" para scripts no firmados.
2. Activar Logging Profundo: Habilitar *Script Block Logging* para capturar el código deobfuscado en memoria.
3. Desplegar EDR con soporte ETW-TI: Garantizar visibilidad sobre la manipulación de registros Dr0-Dr7.
4. Hardenizar Exchange: Configurar el fallback a "Block" y ampliar el buffer de escaneo a 1 M