Ereignisbasierte Systemüberwachung von Windows-PCs

Wissenschaftliche Vertiefung

im Fachgebiet Internet-Sicherheit

Westfälische Hochschule
Gelsenkirchen Bocholt Recklinghausen
University of Applied Sciences

vorgelegt von:	Alexander Schmitz
Studienrichtung:	Technische Informatik
Prüfer:	Prof. Dr. Christian Dietrich

Kurzfassung

Diese Arbeit bietet einen Überblick über das Thema der ereignisbasierten Systemüberwachung von Windows-PCs. Die frühe Erkennung von Angriffen und der Ausführung von Schadsoftware auf Windows-Clients bedingt Sensor-Technologien, die Systemereignisse erkennen und protokollieren. Anhand bestimmter Ereignisse oder komplexerer Ereignismuster können illegitime Vorgänge auf Computersystemen erkannt werden. Der Fokus dieser Arbeit liegt darauf einen detaillierten Überblick über diese ereignisbasierte Sensorik für Windows-Endpunkte zu geben, die Systemereignisse erkennt und Ereignis-Objekte mit beschreibenden Attributen liefert. Im Gegensatz dazu wird die Interpretation der bereitgestellten Ereignisdaten und der Musterabgleich in dieser Arbeit nicht behandelt. Zu Beginn werden dafür die elementaren Aspekte und Begriffe einer solchen Sensor-Technologie allgemein erklärt. Dabei werden einige charakterisierende Grundeigenschaften erläutert, anhand derer solche Senor-Verfahren eingeordnet werden können. Zudem wird auf Basis durchgeführter Messungen eine Abschätzung bezüglich der Auftrittsfrequenzen verschiedener Systemereignisse geliefert. Im Hauptteil werden die zwei ereignisbasierten Monitoring-Systeme Event-Tracing-for-Windows (ETW) und Sysmon vorgestellt. Die Funktionalität, Architektur und interne Funktionsweise der beiden Systeme wird detailliert erläutert. Zudem werden ihre charakterisierenden Basiseigenschaften herausgearbeitet, um Stärken und Schwächen zu erkennen. Der ETW-basierte Sensor und SIGMA-Regelscanner Nextron Aurora, der Sysmon-ähnliche Ereignisdaten bereitstellt, wird ebenfalls behandelt.

Schlüsselwörter

Ablaufverfolgung, Ereignisprotokollierung, Endpunkt-Sensorik, Angriffserkennung, Malware-Erkennung, Prozessverhalten, Systemverhalten, Performance-Counter, PCW, Event-Tracing-for-Windows, ETW, Sysmon, Nextron Aurora, Windows-Kernel

Alexander Schmitz

Abstract

This thesis is an overview of the topic of event based monitoring of Windows PCs. The early detection of cyberthreat attacks and malware execution on Windows clients requires sensor technologies that detect and log system events. Certain events or more complex event patterns can be used to detect illegitimate behavior on computer systems. The focus of this thesis are these event-based sensors for Windows endpoints, which detect system events and provide event objects with descriptive attributes. The interpretation of the provided event data and pattern matching are not covered in this work. At the beginning, the elementary aspects and terms of such a sensor technology are explained in general. This includes some characterizing basic properties, which can used to classify such sensor technologies. In addition, some estimates of the trigger frequencies of various system events based on own measurements are reported. The two event based monitoring systems Event Tracing for Windows (ETW) and Sysmon are presented in the main part. The usage, functionality, architecture and internals of the two systems are explained in detail. Moreover, their most significant characterizing properties are expounded in order to identify advantages and disadvantages. The ETW based sensor and SIGMA rule scanner Nextron Aurora that provides Sysmon like event data is also covered.

Keywords

event tracing, event logging, endpoint sensor, cyberthreat attack detection, malware detection, process behavior, system behavior, performance counter, PCW, event tracing for Windows, ETW, Sysmon, Nextron Aurora, Windows kernel

Inhaltsverzeichnis

1 Ereignisbasierte Systemüberwachung

Aufgrund einer sich vergrößernden Bedrohungslage im Cyberraum wird eine frühzeitige Erkennung von Angriffen auf Computersysteme immer wichtiger. Bei Angriffen auf Organisationen spielen die interaktiven Workstations, an denen die Mitarbeiter arbeiten, oft eine bedeutende Rolle. Auf diesen, als Endpunkte bezeichneten, Client-PCs beginnen oft größere Angriffskampagnen mit einer initialen Infektion. Diese geht in den meisten Fällen von Mail-Anhängen aus, die eine Schadsoftware enthalten. Jedoch gibt es auch viele andere mögliche initiale Infektionsvektoren. Die Angreifer verschaffen sich über Schadsoftware Zugang zu einzelnen Endpunkten und bewegen sich von dort aus über das interne Netzwerk der Organisation zu weiteren Rechnern, die ebenfalls infiziert werden und zu denen sich somit ein illegaler Zugang verschafft wird. Unbemerkt wird so die IT-Landschaft der Organisation von innen heraus kompromittiert. Der Zweck solcher Angriffskampagnen kann sehr unterschiedlich sein. Während bei APT-Angriffen (advanced persistent threat) eine nachrichtendienstliche Informationsgewinnung oder Sabotage im Vordergrund stehen, sind Cybercrime-Angriffe in der Regel finanziell motiviert. Bei letzteren wird die IT der Organisation fundamental gestört, z.B. durch Verschlüsselung sämtlicher Dateien. Für eine Entstörung wird ein Lösegeld von den Akteuren verlangt. Aufgrund der Vielfalt an Vorgehensweisen der Angreifer, unzureichender Sicherheitskonzepte in den Betriebssystemen, Schwierigkeiten IT-Sicherheitskonzepte in Organisationen umzusetzen und unzureichend sensibilisiertes Personal, sind die beschriebenen Angriffe nicht vollständig durch konzeptionelle Sicherheitsmaßnahmen zu verhindern. Stattdessen existieren zum konzeptionellen Schutz alternative, ergänzende Ansätze. Bekannt sind z.B. Anti-Viren-Scanner (AV), die auf Endpunkten installiert, diese statisch nach Schadsoftware durchsuchen. Dabei werden die Inhalte von Dateien und geladenen Abbildern im Speicher mittels Signaturen geprüft, ob diese Muster von bekannter Schadsoftware enthalten. Da Malware sich häufig wandelt, vor einem Angriff bekannt sein muss und auch legitime Software als Angriffswerkzeug (Living-off-the-Land) Verwendung findet, stoßen die rein statischen Erkennungsmethoden an ihre Grenzen. Ein vielversprechenderer weiterer Ansatz schließt das Verhalten von Software in die Erkennung mit ein. Bei diesen dynamischen Erkennungsverfahren wird das Verhalten der Computersysteme permanent beobachtet und seltsame Aktivitäten, die auf einen Angriff hindeuten könnten, erkannt, gemeldet und abgewehrt. Solche Lösungen werden als Endpoint-Detection-and-Response (EDR) bezeichnet. Dabei wird auf den Endpunkten eine Sensor-Software (endpoint agend, EDR agent) installiert, die über verschiedene System-Mechanismen unterschiedliche Systemereignisse erkennt. Die erkannten Ereignisse werden in Form von beschreibenden Ereignis-Objekten protokolliert und vorgefiltert, bevor sie zu einer zentralen externen Stelle übertragen werden [Aar21]. Entweder innerhalb der Organisation oder extern als Dienstleistung besteht ein zentrales Security Operation Center (SOC),

das die Event-Daten der Endpunkte entgegen nimmt, aggregiert und durchsucht. Dies geschieht weitgehend automatisiert durch Systeme, die als Security-Information-and-Event-Management (SIEM) bezeichnet werden. Wurde der Angriff nicht schon auf dem Endpunkt erkannt, soll dieser spätestens im SOC erkannt werden, sodass Gegenmaßnahmen ergriffen werden können. Dabei werden für die Erkennung zwei grundverschiedene Vorgehensweisen kombiniert. Zum einen werden die Ereignisse auf, aus vorherigen Angriffen bekannten, Mustern geprüft. Solches bekannte und als bösartig klassifiziertes Verhalten wird in Form von Regel als Verhaltenssignaturen bekannter Malware und Angriffswerkzeugen beschrieben. Zum anderen wird neben dem Signatur-basierten Scannen der Ereignisdaten zusätzlich eine heuristische Anomalie-Erkennung durchgeführt. Letztere vergleicht das gemessene Systemverhalten mit bekanntem Systemverhalten und soll allgemein Abweichungen erkennen, die auf etwas Bösartiges hindeuten könnte. Ein Beispiel wären hier, das ein auf einmal gemessenes Verhalten, bei dem sehr große Mengen an Dateien gelesen und geschrieben werden, auf eine Verschlüsselungs-Malware hindeuten könnte, die gerade den Rechner verschlüsselt. Das bekannte Systemverhalten mit dem verglichen wird, könnte auf bereits zeitlich zuvor Gesehenem basieren oder auf dem Systemverhalten das zum gleichen Zeitpunkt auf anderen Endpunkten beobachtet wird. Letzter Aspekt verdeutlicht den Vorteil, der aus einer zentralen Verarbeitung der Event-Daten vieler Endpunkte an einer zentralen Stelle entstehen kann.

Eine elementare Komponente, um Angriffe anhand des Verhaltens der Client-PCs zu erkennen, ist das Sensor-System auf dem Endpunkt. Dieses misst über verschiedene Verfahren Abläufe im Computersystem, um bestimmte elementare Systemereignisse zu erkennen und diese mit beschreibenden Attributen zu protokollieren. Diese Arbeit versucht einen Überblick über die verschiedenen Aspekte einer solchen ereignisbasierten Systemüberwachung zu liefern. Da die zu überwachenden Endpunkte in der Regel Windows-PCs sind, handelt es sich bei allen in dieser Arbeit vorgestellten Verfahren um Lösungen für PCs mit Windows-Betriebssystem. Bei der Erklärung der verschiedenen Ereignistypen und Klassen, in denen diese eingeordnet werden können, werden oft zum besseren Verständnis die dahinter liegenden technischen Abläufe im Betriebssystem erklärt. Auch dabei beziehen sich die Beschreibungen stets auf das Windows-Betriebssystem und dessen Kernel.

Jedes sinnvolle Programm muss mit seiner Außenwelt interagieren. Dies gilt auch für Schadsoftware. Die Programme, die in gehosteten Umgebungen (Systeme mit Betriebssystem) ausgeführt werden, treten als ein Prozess auf einem Betriebssystem in Erscheinung. Die Interaktion mit der Außenwelt (Eingabe und Ausgabe nach dem EVA-Prinzip) umfasst in manchen Fällen eine Interaktion mit dem Benutzer. In den meisten Fällen liegt zudem eine Interaktion mit anderen externen Systemressourcen vor. Dies können Dateien, Ressourcen im Internet oder Netzwerk, Konfigurationsstrukturen wie die Windows-Registry oder andere parallel ausgeführte Programme bzw. Prozesse sein. Alle

diese Zugriffe darf das Programm nicht direkt durchführen, sondern muss den Zugriff auf externe Ressourcen beim Betriebssystem-Kernel anfragen, der diese stellvertretend für die Software übernimmt. Die Zugriffe auf externe Ressourcen stellen mit unter den wichtigsten Aspekt des Verhaltens eines Prozesses dar. Dies gilt sowohl für legitime Software, wie auch für Malware in gleichem Maße. Daher liegt es nahe, dass ein Angriff oder allgemeiner schadhaftes Verhalten unter anderem auch anhand des Zugriffsverhaltens einzelner Prozesse auf externe Systemressourcen erkannt werden kann. Aus diesem Grund liegt der Fokus dieser Arbeit vor allem auf der Erkennung des Zugriffsverhaltens von Prozessen auf externe Ressourcen. Allgemein ist bei interaktiven Systemen wie Windows-PCs die Interaktion mit dem Benutzer sehr bedeutend. Einen Großteil der Zugriffe von Prozessen, die eine Benutzerschnittstelle bieten, bezieht sich auf die Interaktion mit dieser und dem dahinter stehenden Benutzer. Schadsoftware und Angreifer versuchen hingegen meist möglichst unbemerkt und verdeckt auf den Systemen zu agieren. Zudem haben die sicherheitskritischen Vorgänge meist nicht mit der Benutzerinteraktion zu tun. Daher kommt im Kontext der Angriffserkennung Zugriffen mit Bezug auf die Benutzerschnittstelle und Nutzerinteraktion keine große Bedeutung zu. Aus diesem Grund wird der Aspekt der Benutzerinteraktion über Benutzerschnittstellen als Teil des zu überwachenden Prozessverhaltens in dieser Arbeit weitestgehend ausgeblendet.

1.1 Ausführungs-Kontext

Ereignisse, die auf einem Computersystem auftreten, besitzen einen Ausführungs-Kontext. Dieser beschreibt die Zuordnung des Ereignisses zu bestimmten System-Ressourcen, in dessen Ausführungs-Kontext das Ereignis stattgefunden hat. Für die meisten Ereignisse kann auf technischer Ebene einem Ereignis-Auftritt ein oder mehrere Prozessor-Kerne, ein oder mehrere Threads sowie ein oder mehrere Prozesse zugeordnet werden. Je niedriger die Abstraktionsebene der Betrachtung ist, desto klarer kann die Zuordnung zu genau einem einzelnen CPU-Kern, Thread und Prozess erfolgen. Je höher die betrachtete Abstraktionsebene der Ereignisse ist, desto stärker müssen Ereignisse mehreren CPU-Kernen, mehrere Threads und mehreren Prozessen zugleich zugeordnet werden. Hinzu kommt, dass die Zuordnung des Ausführungs-Kontextes auf höheren Meta-Ebenen eine immer geringere Aussagekraft besitzt, vor allem bezüglich einzelner Rechenkerne und Threads.

Bei Einzelereignissen, welche direkt einen technischen abgeschlossenen, elementaren Vorgang selbst signalisieren, können Event-Objekte für eine Protokollierung direkt durch den technischen Vorgang erzeugt werden. In diesem Fall, bei dem keine Aggregation von

protokollierten Event-Objekten erfolgen muss, ist die Zuordnung in der Regel eindeutig gegeben. Sie bildet sich aus dem Ausführungs-Kontext der auf dem Rechenkern vorherrschte, in dem Moment als das Ereignis zutage getreten ist. Der CPU-Kern, auf dem das Ereignis ausgelöst wird, ist mit dem Ereignis als Attribut des Ausführungs-Kontextes verknüpft. Zu diesem Zeitpunkt wurde auf dem Rechenkern ein Thread ausgeführt oder war dem Rechenkern zugeordnet. Dieser Thread ist ebenfalls als Teil des Ausführungs-Kontext mit dem Ereignis verknüpft. Der verknüpfte Thread ist Teil eines Prozesses, der auch als Teil des Ausführungs-Kontext mit dem Ereignis verknüpft ist.

Für Ereignisse höher Abstraktionsebenen, welche aus mehreren Ereignissen, niedriger Abstraktionsebenen gebildet werden, kann die beschriebene technische Ermittlung über den Ausführungskontext meist nicht in dieser Form ermittelt werden. Hier könnten die Auftritts-Kontexte der aggregierten Ereignisse vereinigt werden. Jedoch ist zu prüfen, ob für das Ereignis auf einer höheren Meta-Ebene überhaupt eine sinnvolle Information gewonnen wird. Eventuell stellt die Information über beteiligte Threads oder CPU-Kerne für Ereignisse auf dieser Ebene keine Relevanz dar.

Im Falle einer Ereignisprotokollierung können für ein Ereignis, dem einer oder mehrere der genannten Ressourcen zugeordnet werden konnten, diese Ressourcen als Ausführungs-Kontext-Attribute des Ereignisses in die Protokollierung eingehen.

Der im Ausführungs-Kontext zugeordnete Thread und Prozess beschreibt für die meisten Ereignisse den Verursacher des Ereignisses (das Subjekt). Dies trifft jedoch nicht in jedem Fall zu. Ein Beispiel stellen Interrupt-Ereignisse dar. Diese besitzen meist keinen semantischen Bezug zum Thread und Prozess in dessen Ausführungskontext sie auftreten.

Die drei System-Ressourcen-Typen Prozesse, Threads und CPU-Kerne, die eventuell als Ausführungs-Kontext einem Ereignis zugeordnet werden können, sollen im Folgenden näher beschrieben werden.

1.1.1 Prozesse

Auf modernen Computersystemen werden viele Anwendungen und Dienste pseudo-parallel und echt-parallel ausgeführt. Wird ein Programm ausgeführt existiert für die Ausführung ein Prozess. Ein Prozess ist somit ein Programm, das sich in Ausführung befindet. Pro Ausführung existiert dabei ein separater Prozess. Ein Programm kann allgemein auch mehrmals parallel ausgeführt werden. Dies resultiert in mehreren parallelen Prozessen des gleichen Programms.

Die Prozesse stellen dabei eine abstrakte System-Ressource dar, die vom Betriebssystem-Kernel verwaltet wird. Ein Prozess besteht dabei aus einem oder mehreren Threads,

welche die eigentliche Ausführung repräsentieren, einem virtuellen Adressraum, der die Speicherbereiche des Prozesses enthält und Listen mit weiteren für den Prozess vom Kernel bereitgestellten System-Ressourcen [Yos20, S. 3ff][YIRS17, S. 8ff]. Im virtuellen Adressraum befinden sich die Speicherbereiche der Anwendung. Dies sind der Code, die Daten und weitere Ressourcen des Executable-Images (aus EXE-Datei), aus dem der Prozess instanziiert wird, sowie weiterer Speicherabschnitte von Bibliotheks-Images (aus DLL-Dateien) und anderen ladbaren Ressourcen, die der Prozess benötigt [YIRS17, S. 416ff]. Diese Anwendungs-Speicherbereiche liegen in der oberen Hälfte (bei 32 Bit) oder am oberen Rand (bei 64 Bit) des virtuellen Adressraums. Dieser Bereich wird als User-Space bezeichnet. Zudem enthält der virtuelle Adressraum auch alle Speicherbereiche des Betriebssystem-Kernels einschließlich der Bereiche von geladenen Kernel-Modulen bzw. Treibern [YIRS17, S. 401ff]. Diese Speicherbereiche liegen in der unteren Hälfte (bei 32 Bit) oder am unteren Rand (bei 64 Bit) des virtuellen Adressraumes. Dieser Bereich wird als Kernel-Space bezeichnet [YIRS17, S. 397ff][YIRS17, S. 406ff].

Der im Adressraum eines Prozesses befindliche Code setzt sich zusammen aus dem Anwendungs-Code und dem Kernel-Code. Der Anwendungs-Code umfasst die Code-Anteile des geladenen Executables (aus EXE-Datei) sowie der Code-Anteile der geladenen Bibliotheken (aus DLL-Dateien). Auch Teil des Prozesses ist der Kernel-Code, der sich zusammensetzt aus dem Abbild des Betriebssystem-Kerns (aus `ntoskrnl.exe`) und dem Code weiterer geladener Kernel-Module bzw. -Treiber (aus SYS-Dateien). Durch Ressourcenschutz- und Speicherschutz-Mechanismen ist es dem Anwendungs-Code nicht möglich, direkt auf, vom Betriebssystem verwaltete, Ressourcen zuzugreifen. Es können lediglich die Inhalte der Anwendungs-Speicherbereiche (User-Space) des virtuellen Adressraumes gelesen und eventuell geändert werden. Auf die Kernel-Speicherbereiche (Kernel-Space), IO-Geräte, andere Prozesse und abstrakte Organisationselemente wie beispielsweise Dateien und Verzeichnisse kann nicht direkt zugegriffen werden. Demgegenüber steht der Kernel-Code, für den diese Restriktionen nicht gelten. Er besitzt einen Vollzugriff auf den gesamten virtuellen Adressraum und auch auf alle anderen virtuellen Adressräume aller anderen Prozesse. Zudem ist ein Zugriff auf den realen physikalischen Adressraum möglich. Darüber hinaus besitzt der Kernel-Code die Erlaubnis, die Hardware des Rechners zu steuern, was den physischen Zugriff auf IO-Geräte und die Konfiguration der CPU-Kerne und deren Schutzmechanismen mit einschließt. Der Schutz dieser Ressourcen vor der Verwendung durch den Anwendungs-Code wird durch Mechanismen innerhalb eines jeden logischen CPU-Kerns umgesetzt. Die Ausführung kann dabei zwischen zwei Modi wechseln, dem User-Mode, in dem jeglicher Anwendungs-Code ausgeführt wird, und dem Kernel-Mode, in dem der Kernel-Code ausgeführt wird. Das Betriebssystem sorgt dafür, dass beim Einsprung in den Anwendungs-Code in den User-Mode gewechselt wird und somit die Restriktionen aktiv sind. Die Zugriffsfunktionalitäten auf die vom Betriebssystem verwalteten Ressourcen wie z.B. Dateien, Ver-

zeichnisse, Geräte, IPC, Netzwerk-Verbindungen usw. werden dem Anwendungs-Code als Dienstleistung vom Kernel-Code bereitgestellt, der den Zugriff stellvertretend übernimmt und damit die sichere Nutzung der Ressourcen garantiert [YIRS17, S. 26ff]. Möchte der Anwendungs-Code eine dieser Ressourcen verwenden, muss dieser eine Anfrage an den Betriebssystem-Kernel stellen. Dafür ruft der Anwendungs-Code eine System-Funktion des Kernels auf. Dieser Vorgang wird Systemcall genannt und wird durch spezielle Instruktionen der CPU-Kerne umgesetzt. Beim Aufruf wird an eine, vom Kernel beim Bootvorgang konfigurierte, feste Stelle innerhalb des Kernel-Codes gesprungen, die die Anfrage entgegen nimmt und diese abarbeitet. Beim Systemcall wechselt der CPU-Kern automatisch vom User-Mode in den Kernel-Mode. Beim Zurückkehren aus der Funktion wird automatisch von Kernel-Mode zurück in den User-Mode gewechselt. Dafür existieren spezielle Maschinenbefehle (z.B. `SYSENTER` und `SYSEXIT`), die vom Anwendungs-Code beim Tätigen des Systemcalls und vom Kernel-Code beim Verlassen der Systemfunktion aufgerufen werden. Parameter und Rückgabewerte werden nach fest definierten Systemcall-Aufruf-Konventionen über den Stack und/oder CPU-Register übermittelt. Die jeweilige Systemfunktion, die Aufgerufen werden soll, wird in der Regel durch den ersten Parameter des Systemcalls definiert [AIRS21, S. 91ff][Yos20, S. 11ff].

Der Zugriff auf System-Ressourcen gestaltet sich meistens wie folgt. Der Anwendungs-Code beauftragt den Betriebssystem-Kernel mit dem Öffnen einer Ressource. Diese wird daraufhin intern im Kernel mit dem Prozess verknüpft. Dafür wird in die Ressourcen-Tabelle des Prozesses eine Referenz auf die geöffnete Ressource hinzugefügt. Dem Anwendungs-Code wird ein Handle zurückgeliefert. Dieser stellt eine Referenz auf die geöffnete Ressource dar, adressiert also den Eintrag in der Ressourcen-Liste des Prozesses. Mithilfe dieses Handels kann der Anwendungs-Code auf der entsprechenden bereitgestellten Ressource operieren. Alle diese Operationen müssen wiederum über Systemcalls umgesetzt werden, da es niemals einen direkten Zugriff durch den Anwendungs-Code geben darf. Schlussendlich kann der Anwendungsprozess die Ressource wieder freigeben, indem dieser über einen Systemcall das Schließen veranlasst. Die entsprechende Ressource wird daraufhin durch den Kernel freigegeben und der Eintrag in der Ressourcen-Tabelle des Prozesses entfernt [Yos20, S. 15ff][YIRS17, S. 33ff].

1.1.2 Threads

Jeder Prozess besteht aus einem oder mehreren Threads. Ein Thread stellt dabei die kleinste Einheit des Ausführungs-Kontextes dar. Threads sind die Elemente, die auf einem CPU-Kern ausgeführt werden. Ein Thread wird beschrieben durch den Maschinenzustand des CPU-Kerns (Register-Inhalte). Dies beinhaltet die Ausführungsposition im Programmcode, repräsentiert durch den Wert des Instruction-Pointer-Registers, und

den Stack-Kontext, repräsentiert durch die Inhalte der Register Stack-Pointer und Base-Pointer. Das Betriebssystem besitzt zudem eine Reihe von Verwaltungsinformationen bezüglich jedes Threads [YIRS17, S. 20ff][Yos20, S. 8ff][YIRS17, S. 216ff]. In ihnen sind unter anderem die Stacks eines Threads hinterlegt. Im virtuellen Adressraum existieren für jeden Thread zwei Stacks. Einen User-Mode-Stack im User-Space und einen Kernel-Mode-Stack im Kernel-Space. Während ein Thread auf der CPU aktiv ausgeführt wird, arbeitet das Programm auf einem der Stacks, je nachdem, ob es sich im User-Mode oder Kernel-Mode befindet [YIRS17, S. 454ff][Yos20, S. 9ff]. Zudem besitzt ein Thread mehrere Prioritäten, welche sich aus der Prozess-Priorität des Prozesses, dem der Thread angehört, ableiten. Die Ausführungs-Priorität beeinflusst die Reihenfolge, in der Threads auf einem CPU-Kern aktiv ausgeführt werden. Unter Windows wird diese aus den zwei Prioritäten "Base Priority„ und "Dynamic Priority„ gebildet [YIRS17, S. 266ff]. Darüber hinaus gibt es Prioritäten, welche die Rangfolge beim Zugriff auf I/O-Geräte (IO-Priority) und die Zuweisung von Speicher (Memory-Priority) beeinflussen [YIRS17, S. 240ff].

Die Threads der Prozesse werden auf den logischen CPU-Kernen ausgeführt. Der Betriebssystem-Kernel sorgt dafür, den Wechsel von einem Thread zu einem anderen pro logischem CPU-Kern umzusetzen. Dabei wird pro logischem CPU-Kern nur ein Thread aktiv ausgeführt. Die anderen ausführbereiten Threads befinden sich in einer Warteschlange, um bei einem Thread-Wechsel eines CPU-Kerns auf diesem ausgeführt zu werden. Beim Wechsel von einen Thread zu einem anderen, kann implizit ein Prozess-Wechsel stattfinden, wenn die Threads, zwischen denen gewechselt wird, unterschiedlichen Prozessen angehören. In dem Fall muss der Betriebssystem-Kernel unter anderem auch den virtuellen Adressraum ändern [YIRS17, S. 255ff][AIRS21, S. 30ff][YIRS17, S. 286ff]. Die aktive Ausführung eines Threads auf einem CPU-Kern verläuft in den meisten Fällen bis zum Blockieren des Threads. Der Zustand blockiert bedeutet, dass der Thread auf etwas wartet, dass parallel bzw. nebenläufig passiert. Solange gewartet werden muss, kann ein anderer Thread ausgeführt werden. So führt ein Blockieren eines Threads immer zum Wechsel in einen anderen Thread. Solange ein Thread blockiert ist, steht er nicht für die Ausführung auf einem CPU-Kern bereit. Passiert im Kontext eines anderen Threads das Ereignis, auf das der Thread wartet, wird sein Zustand vom Betriebssystem von blockiert auf bereit gewechselt. Ereignisse, auf die Threads warten, sind z.B. das Bereitstellen von Daten durch IO-Geräte (Datenträger-Zugriffe), Eingehen von Netzwerk-Paketen, Freigabe von kritischen Bereichen durch parallel ausgeführte Threads (Mutex), das Ablaufen einer Wartezeit (Timer) [YIRS17, S. 248ff]. Auf interaktiven Systemen warten Threads direkt oder indirekt in den aller meisten Fällen jedoch auf Eingaben vom Benutzer. Auf diesen interaktiven Systemen sind in der Regel die meiste Zeit alle Threads des Systems blockiert. Erst wenn alle Threads blockiert sind, weißt das Betriebssystem einen der Leerlauf-Threads des Leerlauf-Prozesses dem logischen Rechenkern zu. Der Leerlauf-Prozess enthält für jeden Rechenkern des Gesamtsystem einen Leerlauf-Thread. Dieser

besitzt die niedrigste Priorität und wird dadurch nur dann ausgeführt, wenn alle anderen Threads blockiert sind. Der Leerlauf-Thread legt den Rechenkern schlafen, bis zum Eintreten des nächsten Interrupts [YIRS17, S. 292ff]. Das zeitliche Verhältnis von Leerlauf-Thread-Dauer (bzw. Schlafzustand) zu der summierten Thread-Dauern anderer Produktiv-Threads pro Zeit-Intervall ergibt die prozentuale Auslastung eines Rechenkerns [YIRS17, S. 258ff]. Zwar blockieren sich Threads in der Regel selbst oder beenden sich, sodass in den meisten Fällen die faire Ausführung aller bereiten Threads sichergestellt ist, jedoch kann es Fälle geben, in denen Threads überproportional lange arbeiten ohne sich zu blockieren. Um zu gewährleisten, dass alle bereiten Threads gemäß ihrer Priorität Ausführungszeit auf der CPU bekommen, wird die Ausführungsdauer eines Threads vom Betriebssystem-Kernel gemessen. Überschreitet diese einen Schwellwert, wird dem Thread, auch ohne sich selbst zu blockieren, die CPU entzogen, indem ein Wechsel in einen anderen bereiten Thread höherer oder gleicher Priorität stattfindet, wenn einer vorhanden ist [YIRS17, S. 258ff][YIRS17, S. 299ff]. Der Thread wird mit dem Zustand bereit in die Warteschlange überführt, um zu einem späteren Zustand erneut einem Rechenkern zugewiesen zu werden. Kann mehrmals hintereinander kein Thread mit höherer oder gleicher Priorität gefunden werden, setzt Windows die dynamische Priorität des Threads herab [YIRS17, S. 258ff]. So das sichergestellt wird, dass auch Threads niedrigerer Priorität noch aktiv ausgeführt werden. Um die Ausführungszeit des Threads zu bestimmen und den Thread gegebenenfalls zu wechseln, wird der entsprechende Kernel-Code durch ein, in festen Intervallen regelmäßig eintreffendes, Hardware-Interrupt aufgerufen. Zuständig für die Erzeugung dieses Interrupts ist eine elektrische Timer-Schaltung (PIT/RTC-Timer/HPET) im Chipsatz des Mainboards [OSD19]. Der Timer wird durch das Betriebssystem beim Bootvorgang konfiguriert und sein Intervall kann während des Betriebs angepasst werden. Die Periodendauer zwischen zwei Timer-Interrupts liegt zwischen 15,625 ms und 500 µs [AIRS21, S. 66ff]. Es kann beobachtet werden, dass Windows die Frequenz des Timers im Betrieb je nach Auslastung und Anforderung zwischen 66 Hz und 2 KHz in mehreren Stufen variiert. Windows konfiguriert den Interrupt-Controller (APIC) so, dass die Timer-Interrupts abwechselnd von den verschiedenen Rechenkernen bearbeitet werden, sodass die Timer-Interrupt-Rate pro Rechenkern geringer ist. Windows verwendet die Timer-Interrupts für die interne Interrupt-Zeit. Diese ist ein Zeit-Zähler innerhalb der Kernel-Daten, den der Betriebssystem-Kernel verwaltet und der eine von mehreren Zeitbasen darstellt, die das Betriebssystem für Zeitmessungen und Wartevorgänge verwendet[AIRS21, S. 66ff][OSD19][Han19]. Eine sehr wichtige Zeitdauer, die intern bemessen wird, wird als Zeit-Quantum bezeichnet. Dieses Zeit-Quantum beträgt bei aktuellen Windows-64Bit-Systemen 15,625 ms. Der Maximale-Zeitraum, den ein Thread am Stück ausgeführt werden darf, liegt in der Regel (kann geändert werden) bei einem Zeit-Quantum [YIRS17, S. 258ff]. Neben dem beschriebenen Timer-Interrupt können auch andere Interrupts eintreffen. Sie veranlassen einen Einsprung in entsprechende Interrupt-Service-Routinen des Kernel-Codes. Diese ISRs registriert der Kernel

für die verschiedenen Interrupt-Quellen beim Bootvorgang. Für IO-Geräte-Ereignisse ruft die ISR des Kernel entsprechende registrierte ISR-Funktionen der geladenen Gerätetreiber auf. Die Ausführung dieser ISRs und angeschlossener DPCs (deferred procedure call) geschieht im Kontext des Threads, welcher beim Auftreten des Interrupts aktiv ist [AIRS21, S. 32ff][AIRS21, S. 54ff]. Es geschieht kein Wechsel in einen anderen Thread. Beim Eintreten eines Interrupts wird mit dem Einsprung in die ISR des Kernel-Codes in den Kernel-Mode gewechselt, falls dieser nicht bereits aktiv war. Beim Verlassen der ISR benutzt der Kernel-Code einen speziellen Maschinenbefehl (`IRET`), der zurückkehrt an die Code-Stelle, an der die Ausführung unterbrochen wurde. Dabei wird zum User-Mode zurückgewechselt, falls sich die Ausführung zum Zeitpunkt des Auftretens des Interrupts im User-Mode befand [Yos20, S. 159ff][AIRS21, S. 32ff]. Exceptions, die durch Fehlfunktionen in der Ausführung des Codes auftreten, werden in etwa gleicher Weise wie Interrupts behandelt. Die Behandlung von Systemcalls, Interrupts und Exceptions geschieht somit durch den aktiven Thread. Ein Thread-Wechsel findet auch hier nicht statt[AIRS21, S. 85ff].

1.1.3 CPU-Kerne

Wird ein Thread aktiv ausgeführt, geschieht das auf genau einem logischen Rechenkern der CPU. In einem Computersystem können ein oder mehrere Hauptprozessor-Packages verbaut sein. Bei Desktop-Workstations, Laptops und kleineren Geräten handelt es sich in der Regel um ein Package. Ein Hauptprozessor-Package enthält ein oder mehrere physikalische Rechenkerne. Diese physikalischen Rechenkerne können sich auf einem oder mehrerer Mikrochips des Packages befinden. Durch technische Verfahren, die als Simultaneous-Multithreading (SMT) bezeichnet werden, ist es einem physikalischen Rechenkern möglich, mehrere Threads gleichzeitig auszuführen. Bei PC-Prozessoren sind das in der Regel zwei Threads pro physikalischem Rechenkern. Die Simultaneous-Multithreading-Funktion (SMT) ist bei Intel-Prozessoren unter der Produktbezeichnung „Intel Hyper Threading“ (Intel HT) bekannt. Da aus Betriebssystem-Sicht immer einem CPU-Kern ein Thread zugeordnet werden muss, ergibt sich ein neuer Begriff, der „logischer Rechenkern“ genannt wird. Ein physikalischer Rechenkern besteht damit aus mehreren (in der Regel zwei) logischen Rechenkernen [YIRS17, S. 57ff][YIRS17, S. 319ff][YIRS17, S. 461ff]. Im Normalfall führen alle logischen Rechenkerne des Gesamtsystems verschiedene Threads echt-parallel aus. Immer wenn in dieser Arbeit von Prozessoren, CPUs, CPU-Kernen oder Rechenkernen geredet wird, sind in jedem Fall logische CPU-Kerne gemeint.

1.2 Ereignis-Chronologie und Auftrittszeit

Neben dem Ausführungs-Kontext besitzen alle Ereignisse immer eine zeitliche Komponente, die Auftrittszeit des Ereignisses. Beim Protokollieren von Ereignissen, werden Event-Objekte erzeugt und in einem Event-Protokoll gespeichert. Ein Event-Objekt signalisiert den Auftritt eines bestimmten Ereignissen und kann dieses näher beschreiben. Logischerweise sollten Event-Protokolle die zeitliche Komponente der protokollierten Ereignisse widerspiegeln, um im zu einem späteren Zeitpunkt nachzuvollziehen, wann ein Ereignis stattgefunden hat und/oder in welcher Reihenfolge Ereignisse aufgetreten sind. Dies kann mehr oder weniger genau passieren, z.B. indem das Event-Objekt den Auftrittszeitpunkt eines Ereignisses als Attribut enthält, oder ungenauer, indem diese Information nur unzureichend geliefert wird oder fehlt. Auch ohne eine konkrete Zeitangabe in den Event-Objekten können unter Umständen dennoch indirekt zeitliche Informationen transportiert und im Ereignisprotokoll abgelegt werden. Ermöglichen die Datenstrukturen, in denen die Event-Objekte gespeichert werden, wie Puffer oder der Event-Log eine Ordnung bzw. Reihenfolge der gespeicherten Objekte, kann darüber zumindest die chronologische Reihenfolge der aufgetretenen Ereignisse kodiert sein und somit nachvollzogen werden. Für manche Betrachtungen reicht die Information über die Chronologie aus, sodass detaillierte Informationen über konkrete Zeitpunkte nicht notwendig sind. Dies hängt vom jeweiligen Anwendungsfall und Zweck der Ereignisprotokollierung ab. Soll z.B. nur eine Ereignisdetektion ohne Protokollierung stattfinden könnte der genau Auftrittszeitpunkt auch weniger von Bedeutung sein. In den meisten Fällen sind jedoch genaue Zeit-Informationen gewünscht und fast alle Event-Systeme liefern auch Event-Objekte mit konkreten Zeit-Informationen. Dieser Abschnitt erläutert verschiedene Aspekte, die bezüglich der Zeitangaben von Event-Objekten zu beachten sind.

1.2.1 Zeitlich ausgedehnte Vorgänge

Ereignisse auf Computersystemen sind abhängig ihrer Definition nicht immer zeitlich atomar. Z.B. signalisieren und beschreiben Systemereignisse meist Vorgänge auf einem System. So wie die Vorgänge eine zeitliche Dauer besitzen, können somit manche Ereignisse auch eine Dauer beschreiben. Dies kann die zeitliche Dauer des mit dem Ereignis verknüpften Vorgangs sein. Für ein Protokolliertes Event-Objekt, kann somit das Attribut der Auftritts-Zeit ein komplexerer Sachverhalt sein. Ein Event-Objekt könnte z.B. zwei Zeitpunkte aufweisen. Der eine beschreibt den Beginn des Ereignis-Vorgangs, der andere dessen Ende. Statt zwei Zeitpunkte könnte die Kodierung auch durch einen Zeitpunkt und eine Dauer realisiert werden. Alternativ könnten statt Ereignis-Vorgänge mit Zeiträumen zu protokollieren stattdessen nur zeitlich atomare Ereignisse protokolliert werden. Dies wird möglich, indem nicht der Vorgang selbst als Ereignis definiert wird,

sondern das Abschließen oder der Beginn des Vorgangs. So würde es ausreichen, dass das mit dem Ereignis assoziierte Event-Objekt nur einen Zeitpunkt als Attribut besitzt. Es wäre für die Protokollierung und Weiterverarbeitung jedoch wichtig, für jedes protokollierte Ereignis zu wissen, ob es sich beim einzelnen Zeitpunkt um den Beginn, das Ende oder einen anderen Zeitpunkt innerhalb eines ausgedehnten Vorgangs handelt. Natürlich wäre es auch möglich, mehrere Ereignisse pro Vorgang zu protokollieren. So könnten Event-Objekte vor dem Beginn eines Ereignis-Vorgangs erzeugt werden, die den Beginn anzeigen und weitere, die nach dem Abschließen des Vorgangs erzeugt werden, und das Ende anzeigen. Wichtig ist, dass dieser Aspekt beim Umgang mit Event-Quellen im Kontext einer Ereignisprotokollierung für jedes einzelne Event geklärt sein sollte, damit die Ergebnisse richtig interpretiert werden können.

Vorgänge können auch scheitern. Daher sind nur Ereignisse über eine erfolgreich abgeschlossene Aktionen aussagekräftig hinsichtlich eines tatsächlich fertiggestellten Vorgangs. Zeigt ein Ereignis z.B. an, dass ein Prozess beginnt eine bestimmte Aktion durchzuführen, ist es notwendig, auch ein Ereignis zu registrieren, welches das Ende der Aktion signalisiert mit der zusätzlichen Information, ob die Aktion erfolgreich war. Würden nur Ereignisse registriert werden, die das erfolgreiche Ende einer Aktion anzeigen, könnten hingegen gescheiterte Versuche nicht protokolliert werden. In manchen Fällen sind Informationen über gescheiterte Aktionen jedoch auch nützlich und erwünscht. Beispielsweise könnten so auch Zugriffe auf bestimmte Dateien im Kontext eines Angriffes protokolliert werden, die aufgrund von fehlenden Berechtigungen des verursachenden Prozesses scheiterten. In manchen Fällen liefert das Scheitern eines Vorgangs dem Prozess die gewünschte Nutz-Information. Somit kann auch die Protokollierung von gescheiterten Zugriffen für die Überwachung sehr interessant sein. Z.B. versucht ein Prozess eine bestimmte System-Ressource wie beispielsweise eine Datei, einen Registry-Key, einen Mutex usw. zu öffnen, um festzustellen, ob dieser existiert. Ein Nicht-Protokollieren gescheiterter Aktionen diesen Typs würde somit diesen Aspekt des Prozessverhaltens übersehen. Dieses Informationsbedürfnis, zusätzlich auch gescheiterte Aktionen zu registrieren, könnte dadurch bedient werden, dass Aktions-Ende-Ereignisse in jedem Fall zum Erzeugen eines Event-Objektes führen. Der Erfolg oder Misserfolg des Vorgangs könnte über ein Attribut innerhalb des Event-Objektes geliefert werden.

Sieht der Anwendungsfall einer Ereignisdetektion vor, auf Ereignisse zu reagieren, bevor diese passieren, wäre dafür die Detektion von Ereignissen, die den Beginn eines Vorgangs anzeigen, notwendig. Dies könnte z.B. in der Angriffsverhinderung der Fall sein, bei dem z.B. eine ereignisauslösende bösartige Aktion, die ein Prozess direkt vor der Ausführung auslöst, erkannt und darauf reagiert werden, indem der Prozess an der Durchführung gehindert wird. Zusätzlich zur Voraussetzung, dass Event-Objekte vor dem Beginn der Aktion erzeugt werden müssen, muss auch das Event-System eine direkte, synchrone Reaktion einer dritten Instanz auf ein Ereignis erlauben, sodass mit Erzeugen

des Event-Objektes die Ausführung des bösartigen Prozesses unterbrochen wird und die dritte Instanz, die über das Ereignis benachrichtigt wurde, einschreiten kann, bevor der bösartige Prozess die Ausführung fortsetzt.

1.2.2 Zeitliche Unschärfe

Je nach Definition des Ereignisses ergibt sich eine Vorstellung, wann das Auftreten eines bestimmten Ereignisses zeitlich stattfindet und welcher Zeitraum zum Ereignis zählt. Die zeitlichen Parameter bezüglich des Auftretens eines Ereignisses sind im Idealfall somit durch die Definition des Ereignisses eindeutig. Das mit dem Ereignis assoziierte Event-Objekt für eine Protokollierung weicht meist in seinen zeitlichen Attributen jedoch von den realen Zeit-Parametern des Ereignisses ab. Dies ist technisch dadurch begründet, dass die Erkennung des Ereignisses, die Erzeugung und Speicherung des Event-Objektes selbst eine gewisse Zeit in Anspruch nimmt und somit der, in das Event-Objekt eingetragene, Zeitpunkt eine gewisse Latenz zum realen Ereignis aufweist.

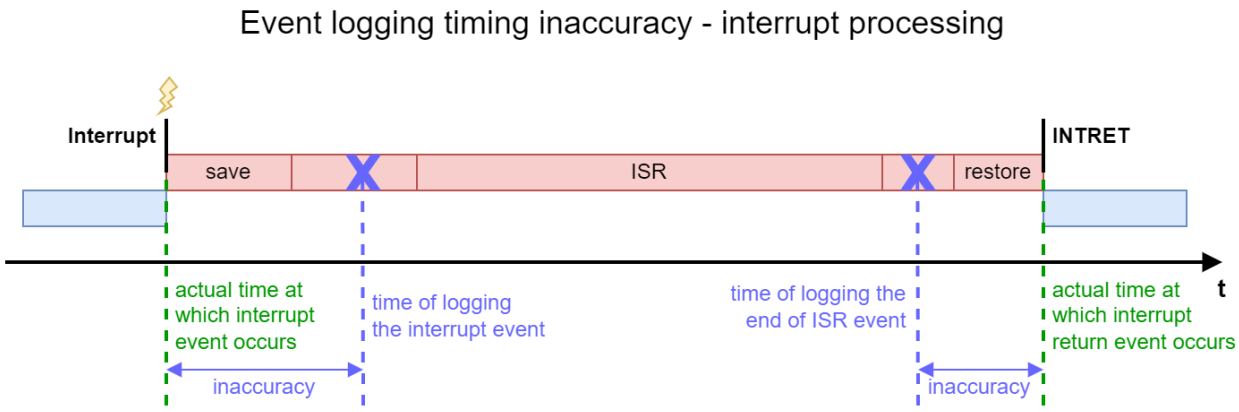

Abbildung 1.1: Zeitliche Ungenauigkeit der Event-Protokollierung am Beispiel Interrupt-Processing

Beispielsweise führt das Ereignis, dass ein Interrupt eintritt, dazu, dass in den Interrupt-Händler des Kernels eingesprungen wird. Dieser muss als erstes notwendige Dinge tun, um den störungsfreien Betrieb zu gewährleisten, wie z.B. Kontext-Sicherungen und Anpassungen bezüglicher der Unterbrechbarkeit (IRQL, Maskierung usw.). Erst danach kann der Kernel-Code veranlassen ein passendes Event-Objekt zu erzeugen, welches das Interrupt-Ereignis signalisiert [AIRS21, S. 32ff][Yos20, S. 37ff]. Wird beim Erzeugen die aktuelle Zeit als Auftritts-Zeitpunkt im Event-Objekt verwendet, liegt dieser Zeitpunkt logischerweise eine gewisse Zeitspanne nach dem eigentlichen realen Ereignis. Was an diesem Beispiel auch sichtbar wird, ist, dass diese Latenz für die meisten Ereignisse sehr klein sollte. Ob dieser Aspekt betrachtet oder vernachlässigt werden kann, hängt vom Anwendungsfall der Ereignisprotokollierung ab. Je nach Größe der Latenz und Auflösung

der zeitlichen Messung, kann es sein, dass der Unterschied zwischen dem Zeitpunkt eines realen Ereignisses und dem protokollierten Zeitpunkt im Event-Objekt, auf der Auflösungsskala der zeitlichen Attribute des Event-Objektes nicht sichtbar wird.

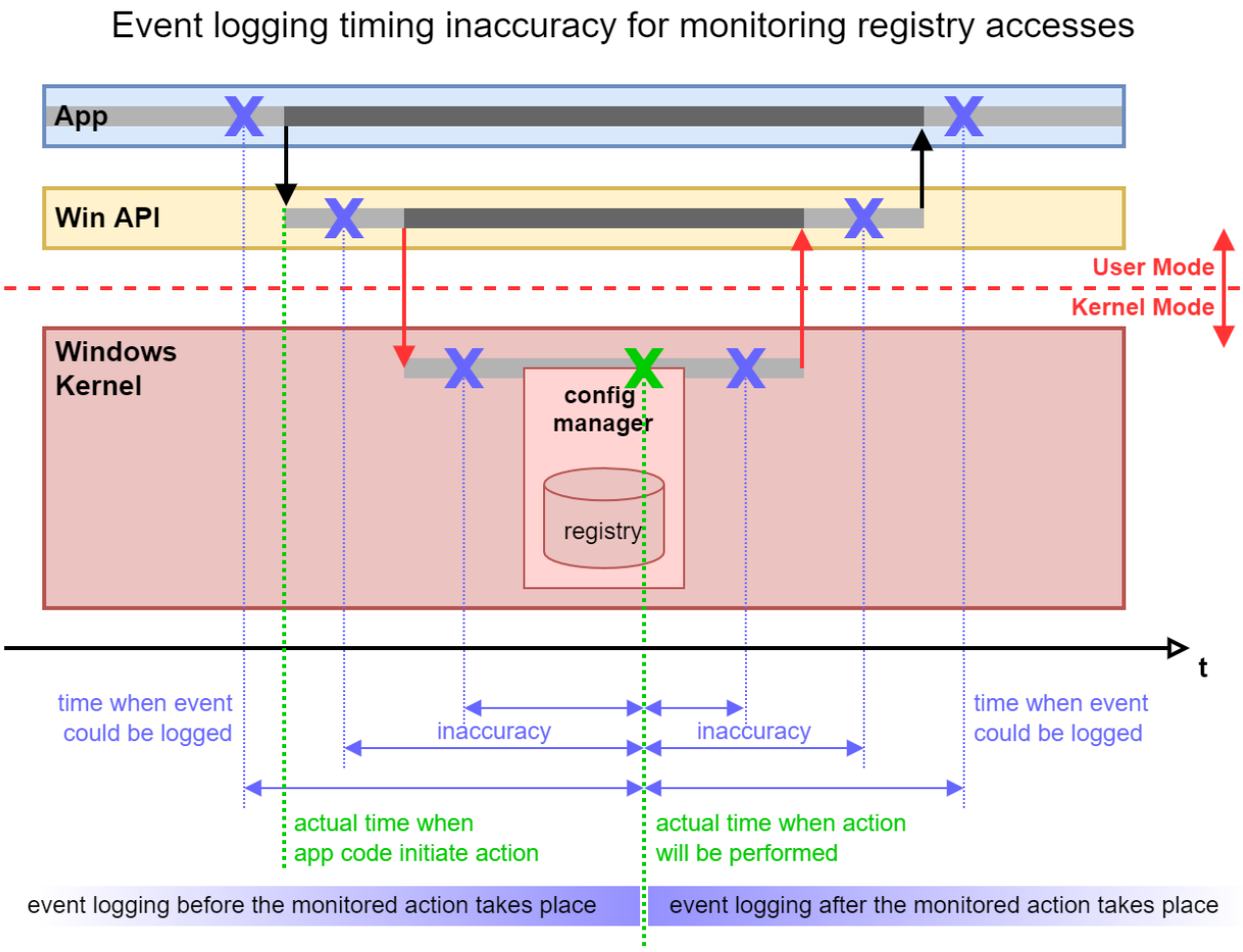

Abbildung 1.2: Zeitliche Ungenauigkeit der Event-Protokollierung am Beispiel Registry-Zugriff

Je nachdem, wie der Beginn oder das Auftreten eines Ereignisses definiert ist, kann eine solche Unschärfe auch für Aktions-Ereignisse bezüglich Zugriffen von Prozessen gefunden werden. Die Frage die sich stellt, ist, welcher Zeitpunkt ist als Anfang einer Aktion definiert. Hier stellt sich die Frage, beginnt der Zugriff z.B. auf eine Datei damit, dass der Anwendungs-Code eines Prozess entsprechende Funktionen der Laufzeitumgebung aufruft, in dem Moment, indem der erste notwendige Systemcall getätigt wurde oder dann wenn tatsächlich Änderungen an den Strukturen im Dateisystem vorgenommen werden. Weiter ist offen, ob der Kernel-Code des Datei-Managers das Event-Objekt für das Beginn-Ereignis erzeugt. So läge der Zeitpunkt dieser Erzeugung eindeutig hinter den zwei zuerst genannten Beginn-Zeitpunkt-Varianten des Ereignisses.

1.2.3 Zeitliche Auflösung

Um in den Event-Objekten Zeitangaben als Attribute zu speichern, müssen diese technisch ermittelt werden. Dafür kommen Zeitquellen zum Einsatz. Diese sind unter anderem durch ihre Messgenaugkeit und ihre Auflösung charakterisiert [Mic4a][Han19]. Unabhängig der Quelle werden die Zeitpunkte oder Zeiträume in bestimmten Datenformaten innerhalb von Event-Objekten gespeichert. Auch durch dieses Format kann eine Einschränkung bezüglich der Auflösung der gespeicherten Messwerte bestehen. Die jeweils gröbere Auflösung zwischen der verwendeten Quelle und dem verwendeten Format bestimmen als limitierender Faktor die zeitliche Auflösungsgenaugigkeit eines Zeit-Attributes eines Event-Objektes.

Die Zeitquellen können bezüglich ihrer Auflösung sehr verschieden ausfallen. Hochfrequenz-Zähler innerhalb der CPU-Kerne (bei PCs innerhalb der Local-APICs) weisen Auflösungen im Sub-Nanosekundenbereich auf und sind in der Lage die zeitliche Dauer der Ausführung einzelner Maschinen-Instruktionen zu messen. Auch Taktzyklen selbst können als Zeitquelle in Betracht kommen. Sie werden ähnlich wie die Werte der Hochfrequenz-Zähler über spezielle Maschinenbefehle aus Zählregistern (bei PCs innerhalb der Local-APICs) des Rechenkerns ermittelt. Jedoch stellen Taktzyklenzähler in modernen PCs keine stabile Zeitquelle dar, da die Taktfrequenz der CPU-Kerne nicht einheitlich sein muss und zudem lastbedingt variiert wird. Neben Hochfrequenz-Zählern existieren in modernen PCs noch viele weitere Timer-Schaltungen im Chipsatz auf dem Mainboard, deren Interrupts als Taktgeber für Software-Zeitquellen in Frage kommen. Dies sind z.B. der Programmable-Interval-Timer (PIT), der Timer in der Echtzeit-Uhr (RTC-Timer), der High-Precision-Event-Timer (HPET) oder der ACPI-Power-Management-Timer (ACPI-PM-Timer). Auf Betriebssystem-Ebene ist z.B. die Interrupt-Zeit [Mic4c][AIRS21, S. 66ff] oder das Zeit-Quantum [YIRS17, S. 258ff] als Zeitquellen zu nennen, die auf Timer-Interrupts (häufig PIT) basieren. Die Auflösung der Interrupt-Zeit ist mit 15 ms bis 500 µs unter Windows jedoch für die Messung hochfrequente Ereignisse zu gering. Neben den Timern besitzen Computersysteme Echtzeit-Uhren (RTC), welche die konfigurierte Systemzeit einschließlich des Datums liefern. Auch wenn der Sub-Sekundenbereich meist nicht vom Benutzer eingestellt werden kann, besitzen diese Zeitgeber eine Genauigkeit in den Mikrosekundenbereich hinein. Für Zeitmessung und Zeit-Attribute in Event-Objekten verwendet das Betriebssystem und viele Event-Systeme meist eine Kombination aus verschiedenen der genannten Zeitquellen, die in einem aggregierten Format zusammengefasst werden und Auftrittszeiten in einer gewissen finalen Auflösung genau beschreiben [Mic5b][Mic4a].

1.2.4 Zeitbasis

Um Zeit-Attribute protokollierter Event-Objekte korrekt zu interpretieren, ist neben dem Format und der Auflösung auch die Zeitbasis entscheidend. Technisch gesehen beinhaltet die Frage bezüglich der Zeitbasis meist auch die beim erstellen des Event-Objektes genutzte Zeitquelle. Zeitbasen könnten z.B. die durch den Benutzer konfigurierte Systemzeit sein. Denkbar wären aber auch die vom Kernel ermittelten Ausführungszeiten von Prozessen oder Threads. Diese existieren für jeden Prozess und Thread für die Lebensdauer und beginnen bei dessen Erzeugung mit Null. Alternativ könnte auch die Prozessor-Zeit genutzt werden. In der Regel handelt es sich dabei aber um mehrere Zeitbasen. Es existiert meist eine Prozessor-Zeit für jeden Rechenkern, die bei dessen Initialisierung mit Null beginnt zu zählen und nach einer gewissen Zeit durch einen Überlauf wieder auf Null zurückfällt. Die Komplexität bezüglich Zeitmessungen und der Bestimmung von Zeitpunkten von Vorgängen innerhalb Computersystemen soll nicht weiter vertieft werden. Wichtig ist jedoch, dass dieser Aspekt berücksichtigt werden muss. Dazu ist es notwendig die Zeitbasis, auf der Zeitangaben innerhalb der Attribute eines Event-Objektes basieren, zu kennen.

Unter Umständen kann es sein das Event-Objekte verschiedener Quellen eines Computer-Systems verschiedene Zeitbasen nutzen. Sobald diese innerhalb einer Ereignisprotokollierung oder anschließenden Weiterverarbeitung in Beziehung gebracht werden sollen, kann dies zu Problemen führen. Hierfür ist es notwendig, die Zeitangaben in ein einheitliches Format und auf eine einheitliche Zeitbasis zu überführen. Gegebenenfalls ist dafür die Synchronisation bestimmter Messverfahren und Zeitquellen notwendig. Dieses Problem tritt im besonderen Maße dann auf, wenn zwischen protokollierten Ereignissen verschiedener Computersysteme eine zeitliche Beziehung hergestellt werden soll.

1.3 Abstraktionsebenen der Ereignisse

Ereignisse zur Ablauf- oder Verhaltensverfolgung von Software oder Computersystemen als ganzes können auf verschiedenen Abstraktionsebenen entstehen. Dies impliziert, dass Ereignisse auf semantisch höheren Ebenen mehrere Ereignisse auf semantisch niedrigeren Ebenen einschließen können.

Im Folgenden werden beispielhaft vier besonders charakteristische Abstraktionsebenen beschrieben, in der Reihenfolge von feingranularer Ablaufverfolgung der Softwareausführung bis hin zu semantischen Ereignissen höher Abstraktionsebenen. Dabei ist zu berücksichtigen, dass die Aufteilung in die hier gewählten vier Ebenen exemplarisch zu verstehen ist. Sie sollen beispielhaft dazu dienen, den Grund-Sachverhalt der Betrachtungsebenen näher zu bringen. So können die hier präsentierten Ebenen beliebig weiter zergliedert sowie weitere nicht berücksichtige Ebenen gefunden werden.

1.3.1 Ausführungsebene

Die Ausführungsebene stellt eine feingranulare Ebene der Ablaufverfolgung der Code-Ausführung dar. Ereignisse auf dieser Ebene können z.B. die Nutzung bestimmter Maschinenbefehle sein, das Durchlaufen markanter Code-Abschnitte, Sprünge im Maschinencode oder das Einspringen in und Rückkehren aus bestimmten Prozeduren. Letzteres kann beispielsweise für die Erstellung von Aufruf-Zeit-Diagrammen, die den Aufrufstack über die Zeit protokollieren (sog. Flame-Graph), genutzt werden. Auch die Signalisierung von Wechseln der Ausführung zwischen dem Code verschiedener Module bzw. Bibliotheken (Inter-Modular-Transition) stellt ein etwas gröberes Ereignis auf dieser Abstraktionsebene dar.

Die ausgelösten Ereignisse entstehen durch die Ausführung von Code im Kontext eines Threads und Prozesses und können somit eindeutig einem Thread und einem Prozess zugeordnet werden, vorausgesetzt die Betrachtung bezieht sich nicht auf frühen Boot-Code, bei dem noch nicht zwischen verschiedenen Tasks unterschieden wird. Gleichwohl können die Ereignisse auch einem Rechenkern zugeordnet werden, auf dem der Thread zum Zeitpunkt des Auslösens des Ereignissen ausgeführt wurde. Letztere Zuordnung kann für Spezialfälle interessant sein, z.B. im Kontext von Performance-Profiling in Zusammenhang mit Non-Uniform-Memory-Access (NUMA). Die Performance kann aufgrund der Topologie von Cache- und Speicherstrukturen sowie von der Vorgeschichte durch die Wahl des Rechenkerns, auf dem ein bestimmter Thread ausgeführt wird, unterschiedlich ausfallen.

Zu all diesen Ereignissen ist zu sagen, dass diese allgemein standardmäßig nicht protokolliert werden. Dies ist verständlich, da ihre Auftrittsfrequenz sehr hoch und die Zeit

zwischen Ereignissen dieser Ebene recht klein ist. Eine Protokollierung würde daher zu Latenzen in der Ausführung führen mit einem großen Overhead an zusätzlichem Rechenaufwand für die Protokollierung im Vergleich zur beobachteten Ausführung des Nutz-Codes. Zudem können auf einer so fein-granularen und detaillierten Betrachtungsebene extrem große Mengen an Ereignis-Daten anfallen.

Für Debugging-, Profiling- und Analyse-Sitzungen kann eine ereignisbasierte Ablaufverfolgung auf Ausführungsebene dennoch sehr nützlich sein. Umgesetzt wird die Ereignisgenerierung auf dieser Ebene durch verschiedene Werkzeuge der Instrumentierung. Einige davon benutzen Hardware-Funktionen der CPU wie z.B. Intel Processor Trace (Intel PT), mit welchem unter anderem Sprünge im Maschinencode und Befehlssequenzen protokolliert werden können. Andere Methoden sind softwarebasiert und nutzen Dynamic-Binary-Instrumentation oder Hooking. Je nach Grad der Einwirkung ist aufgrund des Overheads die Ausführung instrumentierter Software jedoch deutlich langsamer. Daher eignen sich solche Methoden nicht für den Monitoring-Einsatz auf Produktivsystemen.

1.3.2 Systemebene

Die Systemebene ist eine Stufe gröber als die Ausführungsebene. Sie gruppiert elementare Ablauf-Ereignisse eines Betriebssystems. Dies sind Ereignisse, die Systemcalls, Kontextwechsel, Interrupts, Exceptions, Page-Faults usw. anzeigen. Auch spezielle Mechanismen der verzögerten oder asynchronen Ausführung wie beispielsweise unter Windows die Deferred-Procedure-Calls (DPCs) oder Asynchrone-Procedure-Calls (APCs) zählen zu Vorgängen, dessen Ereignisse dieser Abstraktionsebene zugeordnet werden können. Dabei gibt es Ereignisse, die sich einem konkreten Thread und Prozess zuordnen lassen und andere die allgemein mehr oder weniger als systemweite Ereignisse angesehen werden können. Systemcalls, Exceptions einschließlich Page-Faults werden beispielsweise absichtlich oder unabsichtlich durch die Ausführung des Codes eines spezielle Threads verursacht. Dahingegen geschehen Interrupts in der Regel unabhängig vom Verhalten eines Threads. Sie treten zwar im Kontext eines laufenden Threads und Prozesses auf, jedoch besteht meist keine semantische Verbindung zwischen ihnen und dem zu diesem Zeitpunkt ausgeführten Thread. Hier wäre im Gegensatz eher eine Verbindung zum CPU-Kern, auf dem ein Interrupt eingegangen ist, herzustellen. Ähnlich ist es mit Deferred-Procedure-Calls (DPCs), welche unter anderem durch den Code von Interrupt-Service-Routinen (ISRs) angelegt werden. Auch ihr Zweck lässt sich eher mit dem logischen Prozessor-Kern in Verbindung bringen als mit dem Thread und Prozess in dessen Kontext die Prozedur ausgeführt wird. Ereignisse auf Systemebene können somit immer einem Thread- und Prozess-Kontext sowie einem Rechenkern zugeordnet werden. Welche Betrachtung die semantisch aussagekräftigere ist, hängt vom Typ des Systemereignisses ab.

Der Windows-Betriebssystem-Kernel zählt und protokolliert Ereignisse dieser Abstraktionsebene. Über die Performance-Counters-for-Windows (PCW) können die Auftrittsfrequenzen der genannten Ereignisse in Events-pro-Sekunde abgefragt werden. Das in dieser Arbeit detailliert beschriebene Event-Tracing-for-Windows (ETW) ermöglicht zudem, die genannten Ereignisse inklusive Parametern zu protokollieren.

Die Auftrittsfrequenzen der Ereignisse auf dieser Ebene können für einige Ereignistypen relativ hoch sein, wenn auch deutlich geringer als die Auftrittsfrequenzen der Ereignisse der Ausführungsebene (genauere Informationen dazu finden sich im Kapitel Auftrittsfrequenzen). Daher stellt eine Protokollierung auf dieser Ebene ebenfalls einen Overhead dar, der das System verlangsamen kann. Zudem entstehen bei einer solchen Protokollierung in kurzer Zeit große Mengen an Daten. Dies ist insbesondere dann der Fall, wenn eine dauerhaft laufendes Monitoring der Ereignisse durchgeführt werden soll.

1.3.3 Prozess/Thread-Verhaltensebene

Das Verhalten eines Prozesses kann maßgeblich durch sein Zugriffsverhalten auf Strukturen der „Außenwelt“ beschrieben werden. Mit dem Begriff „Außenwelt“ sind in dem Fall alle vom Betriebssystem verwalteten Ressourcen gemeint, mit denen der Prozess interagiert. Dies können beispielsweise Dateien und Verzeichnisse auf Datenträgern, Registry-Einträge, andere Prozesse und Dienste, I/O-Geräte, Netzwerk-Sockets und IPC-Strukturen wie z.B. Mutexe, Semaphoren und Pipes sein. Mit all diesen durch das Betriebssystem verwalteten Ressourcen können Prozesse interagieren, in dem sie auf die jeweiligen System-Objekte in verschiedenster Weise zugreifen. Das Verhalten von Prozessen wird also unter anderem hauptsächlich durch Zugriffe auf System-Objekte beschrieben. Diese Zugriffe gliedern sich in elementare Einzelaktionen, die konkret von den Threads eines Prozesses durchgeführt werden. Dabei werden die Ressourcen in der Regel jedoch nicht dem durchführenden Thread sondern dem Prozess als ganzes bereitgestellt. Jede Einzel-Interaktion wird jedoch von einem konkreten Thread des Prozesses durchgeführt und kann diesem eindeutig zugeordnet werden. So kann beispielsweise ein Thread eine Datei öffnen, während ein anderer Thread danach in die geöffnete Datei schreibt. Die jeweilige Ressource wird dem Prozess über einen Handle durch den Betriebssystem-Kernel bereitgestellt. Das Öffnen einer Datei veranlasst das Bereitstellen der Ressource mit dem Zugriffs-Handle durch das Betriebssystem. Das Schreiben in die Datei stellt einen Verwendungsvorgang der bereitgestellten Ressource dar. Mit anderen Ressourcen wie z.B. Registry-Keys, Prozessen, Threads, IPC-Strukturen, Sockets oder IO-Geräten verhält es sich in ähnlicher Weise. Die Zuordnung der Einzelereignisse zu einem speziellen Rechenkern kann nicht in jedem Fall eindeutig geschehen, da Threads während der Durchführung einer ereignisauslösenden Aktion den Rechenkern wechseln können. Dies kann dann passieren, wenn der Thread innerhalb der Aktion blockiert und bei der

späteren Reaktivierung einem anderen Rechenkern zugeordnet wird. Zudem ist die Information bezüglich des ausführenden CPU-Kerns auf dieser Ebene der Betrachtung meist nicht relevant.

Auch an dieser Stelle ist zu erkennen, dass Ereignisse der Verhaltensebene Ereignisse niedrigerer Abstraktionsebenen einschließen. Konkret wird die Interaktion eines Threads oder Prozesses mit genannten Systemressourcen (der Außenwelt) durch Code des Betriebssystem-Kernels umgesetzt. Diese Funktionalität des Kernels wird durch Bibliotheken gekapselt vom Anwendungs-Code über Systemcalls angefragt. Der beim Systemcall stattfinde Wechsel vom User-Mode in den Kernel-Mode sowie das Zurückwechseln beim Abschließen der Anfrage erzeugt Systemcall-Ereignisse auf der Systemebene. In gleicher Weise wie bei den Systemereignissen zählt und protokolliert der Windows-Kernel auch die meisten in diesem Abschnitt beschriebenen Ereignisse und stellt die Auftrittsfrequenzen über die Windows-Performance-Counters (PCW) und die Ereignisprotokollierung über das Event-Tracing-for-Windows (ETW) bereit.

Die Auftrittsfrequenzen und Ereignismengen auf dieser Abstraktionsebene variieren stark hinsichtlich des betrachteten Ereignistyps und der Software, die ausgeführt wird. Werden Ereignisse nur für einen oder eine kleine Menge an Threads oder Prozessen gesammelt, sind diese handhabbar. Wird eine solche Ereignisprotokollierung jedoch für alle Prozesse des gesamten Systems durchgeführt, kann die Menge je nach Systemauslastung groß sein. Sie ist aber in der Regel deutlich geringer, als die Ereignismenge der darunter angesiedelten Abstraktionsebenen.

1.3.4 Aggregierte Prozess/Thread-Verhaltensebene

Wie beschrieben, bestehen Zugriffe auf vom Betriebssystem verwaltete Ressourcen meist aus einer Folge von mehreren Einzelereignissen. Beispielsweise beinhaltet das Ändern eines Wertes eines bestimmten Feldes eines Registrierungsschlüssels mehrere Einzelereignisse, wie das Öffnen des Registrierungsschlüssels, das Setzen des Wertes eines spezifischen Feldes und das abschließende Schließen des Registrierungsschlüssels. Das „Öffnen“-Ereignis liefert den Schlüssel-Namen bzw. Pfad, wohingegen das „Wert-Ändern“-Ereignis den Index und/oder Bezeichner des Feldes und den neuen Wert als Parameter liefert. Abhängig vom Zweck der Ereignisprotokollierung kann es wünschenswert sein, eine gröbere oder höhere Abstraktion des Vorgang als ein Ereignis zu protokollieren. So könnten die beschriebenen drei Ereignisse zu einem Registry-Wert-Ändern-Ereignis zusammengefasst werden, welches sowohl den Schlüsselnamen bzw. den Pfad sowie den Feld-Bezeichner und den geschriebenen Wert als Parameter enthält. Solche Ereignisse, die einen zusammenhängenden semantischen Vorgang des Prozessverhaltens als Ganzes

beschreiben, werden hier in dieser Einteilung der Abstraktionsebenen als Ereignisse der „Aggregierten Verhaltensebene“ bezeichnet.

Besonders plakativ wird der Unterschied der zwei Abstraktionsebenen der Verhaltensprotokollierung bei der Betrachtung von Gesamtvorgängen, die einen Stapel von Funktionalitätsschichten durchlaufen, wobei auf jeder Funktionsschicht mehrere Einzelereignisse spezifiziert werden können. Dies ist im besonderen Fall bei der Netzwerk-Kommunikation zu finden, die durch den TCP/IP-Protokoll-Stapel implementiert ist. Ein aggregierter Gesamtvorgang wie das Empfangen einer HTTP-Antwort von einem Server, teilt sich in einen TCP-Verbindungsaufbau-Vorgang, das Empfangen von Daten-Segmenten, das Senden von Bestätigungen und den TCP-Verbindungsabbau-Vorgang auf. Hinzu kommen weitere Einzelereignisse auf tieferen Ebenen. So resultieren die meisten TCP-Ereignisse in einem oder mehreren Sende- und Empfangsereignissen auf der darunterliegende IP-Schicht. Werden auch weitere darunterliegende Schichten betrachtet wie zum Beispiel das Lesen und Schreiben von Ethernet-Frames auf physikalische Netzwerk-Medien steigt die Anzahl der Einzelereignisse weiter an. Das Beispiel Netzwerk/Internet-Kommunikation zeigt bezüglich Aggregation jedoch noch eine weitere Dimension auf. Nicht nur vertikal durch die Schichten des Protokollstapels entstehen Ereignisse, sondern auch horizontal in der Abfolge. So geht in manchen Fällen einer Anfrage auf Anwendungsschicht eine Namensauflösung durch den DNS-Dienst oder eine Zertifikatsprüfung durch den TLS-Dienst voraus. Auch diese beinhalten bezüglich des Prozessverhaltens Einzelereignisse und eventuell wiederum weiterer Einzelereignisse tiefer liegender Schichten (z.B. wenn eine DNS-Abfrage über das Netzwerk getätigt wird). Je nach Zweck der Ereignisprotokollierung ist es auch hier meist ausreichend für die Verhaltensüberwachung eines Prozesses, gröbere Ereignisse, wie beispielsweise HTTP-Anfragen und Antworten in aggregierter Form zu erhalten, in denen die Ereignisse und Parameter aller involvierten Vorgänge bereits integriert sind.

Einige Ereignisse auf dieser Abstraktionsebene können nicht in jedem Fall einem Thread zugeordnet werden, da es möglich ist, dass verschiedene Threads eines Prozesses am aggregierten Gesamtvorgang beteiligt sein können. In der Regel kann jedoch das Ereignis, welches den Gesamtvorgang beschreibt, weiterhin einem Prozess eindeutig zugeordnet werden. Eine Zuordnung zu einem Rechenkern ist auf dieser Abstraktionsebene meist nicht sinnvoll und auch oft nicht widerspruchsfrei möglich.

Die Ereignisse können aus der Aggregation von Ereignissen der Verhaltensebene erzeugt werden. Ein solches Vorgehen verringert die Menge an Ereignissen und die Auftrittsfrequenzen. Zudem werden die Einzelereignisse der Verhaltensebene in sinnvolle und semantisch nachvollziehbare Beziehungen zueinander gebracht. In Abhängigkeit des Zwecks der Ereignisprotokollierung reicht die Detailtiefe dieser Abstraktionsebene aus, während zusätzliche Vorteile auf Seiten der Performance und der Handhabbarkeit entstehen. So

kann je nach Anwendungsfall die Speicherung, Weiterverarbeitung und Analyse einer geringeren Menge aggregierter und aussagekräftigerer Ereignisse von Vorteil sein. Für den Anwendungsfall der Angriffserkennung auf Basis von Ereignisprotokollierung erscheint dies sehr sinnvoll.

Solche aggregierten Verhaltensereignisse werden meist als Output von Logging-Systemen oder im Kontext der Angriffserkennung als Output von Endpoint-Sensorik genutzt. Im letztgenannten Anwendungsfall sollte Sysmon erwähnt werden. Sysmon ist ein kostenfreies Werkzeug der Microsoft Sysinternals Suite (Mark Russinovich, David Solomon, Alex Ionescu), welches einige aggregierte Verhaltensereignissen erzeugt und diese in den Windows-Event-Log einträgt. Sysmon nutzt dazu intern zum einen die Aggregation von ETW-Ereignissen, sowie weitere selbst implementierte Monitoring-Funktionalitäten auf Basis von Kernel-Callbacks und Filter-Treibern, die Sysmon über einen eigenen Kernel-Mode-Treiber in den Windows-Kernel einbringt. Die Abdeckung der Verhaltensvorgänge eines Prozesses, die Sysmon protokolliert, ist dabei jedoch nicht vollständig. Die Software konzentriert sich auf die relevantesten Verhaltensvorgänge um Schadsoftware und Angriffe zu erkennen. Beispielsweise wird das Schreiben in eine Datei protokolliert, das Lesen aus einer Datei jedoch nicht. Für die allgemeine Angriffserkennung kann dies unter Umständen ausreichend sein, für die möglichst lückenlose Überwachung des Verhaltens eines Prozesses jedoch nicht.

1.3.5 Höher aggregierte Ereignisse

Abstraktionen lassen sich theoretisch beliebig weiter erhöhen. Höhere Abstraktionsebenen, die auch als Meta-Ebenen bezeichnet werden können, beinhalten Ereignisse mit einem höheren Grad an Semantik. Solche Ereignisse beschreiben den Zweck mehrerer Ereignisse der darunterliegenden Verhaltensebene. Sie adressieren die Frage, aus welchem Grund ein Prozess bestimmte Vorgänge durchführt. Daraus resultieren wiederum Verhaltensereignisse jedoch auf einem sehr viel gröberen und zugleich noch viel aussagekräftigeren Niveau. Dies ist jedoch nur dann der Fall, wenn die Ereignisse auf dieser Ebene wirklich korrekt erzeugt bzw. Ereignisse der tiefer liegenden Ebenen korrekt aggregiert und interpretiert werden.

Beispiele wären hier Ereignisse die anzeigen, dass ein Prozess eine Software installiert oder aktualisiert. Im Bezug auf Schadsoftware und Angriffserkennung könnten dies Ereignisse wie Persistierung, Thread-Injection, Exploitation, C2-Kommunikation, Spreading usw. sein, die von einem bösartigen Malware-Prozess oder einem entführten Anwendungs-Prozess ausgelöst werden.

Die Zuordnung von Ereignissen auf dieser Abstraktionsebene zu einzelnen Rechenkernen des Computersystems ist in der Regel nicht möglich und auch meist nicht sinnvoll. Auch

können Ereignisse meist nicht mehr einem einzelnen verursachenden Thread zugeordnet werden, da oft mehrere Threads beteiligt sind. Dies kann auch für die Zuordnung zu Prozessen gelten. An einigen Ereignissen sind unter Umständen mehrere Prozesse beteiligt. Letzteres ist vor allem vom Grad der Abstraktion abhängig. Konkret spielt hier der Fokus, ob die Ereignisse das Verhalten von Prozessen oder mehr das Gesamtverhalten des Computersystems beschreiben, eine Rolle.

Für den Anwendungsfall der Angriffserkennung werden solche beschriebenen höheren Aggregationen von EDR-Lösungen durchgeführt. Dabei werden sowohl regelbasierte Verfahren, wie Verfahren der Künstlichen-Intelligenz bzw. des Machine-Learning eingesetzt. Die Erkennung und Zuordnung von Ereignissen findet dabei hauptsächlich auf der Basis zuvor gesehener Verhaltensmuster statt. Diese wurden in Form von Verhaltenssignaturen entweder automatisch erlernt oder mit Expertenwissen dem System hinzugefügt (z.B. über SIGMA-Signaturen). Die Verarbeitung findet zum Teil auf den Endpunkten, wie auch in zentralen SIEM-Systemen (meist Cloud-basiert) statt, zu denen die Event-Objekte übermittelt werden. Die Aggregation zu semantisch höheren Ereignissen mit zusätzlicher Filterung auf den Endpunkten reduziert die Anzahl an Ereignissen deutlich und macht den Overhead für die Weiterverarbeitung handhabbarer. So verbessert sich durch eine geringere Datenmenge und Frequenz der Overhead und die Netzlast in der Kommunikation mit zentralen SIEM-Systemen zusätzlich. Dem gegenüber steht, dass für das Erlernen oder Erkennen neuer Angriffsmuster auch weniger aggregierte Rohdaten der darunterliegenden Abstraktionsebenen benötigt werden. Da das Erzeugen zu erkennender Angriffsmuster aus den Ereignisdaten meist im SIEM passiert und davon profitiert, dass dabei Ereignisdaten von sehr vielen verschiedenen Endpunkten miteinbezogen werden, sollten weniger aggregierte Ereignisse auch an das SIEM weitergeleitet werden. Diese genannten Aspekte stehen im Widerspruch, sodass hier ein Kompromiss zwischen Overhead, Netzlast und Funktionalität getroffen werden muss.

Die Abstraktion kann theoretisch beliebig erweitert werden bis hin zum Erkennen von Phasen einer Angriffskampagne auf Basis großer Mengen von Ereignis-Protokolldaten über längere Zeiträume. Letzteres bezieht sich doch weniger auf das Verhalten einzelner Prozesse eines Computersystems als vielmehr auf viele Prozesse und Ressourcen die über mehrere Rechner einer Netzwerk-Infrastruktur verteilt sind. Aggregationen auf dieser Ebene sind aktuell eher theoretisch anzusehen, als dass dafür aktuell technische Lösungen bereitstehen. Das Thema ist aktuell jedoch Bestandteil der Forschung.

1.4 Subjekt-Objekt-Beziehungen

Manche Ereignisse beschreiben Aktionen, die als Subjekt-Objekt-Beziehung modelliert werden können. Dabei stellen diese Ereignisse eine Subjekt-Objekt-Beziehung zwischen zwei System-Ressourcen dar. Das Subjekt ist die System-Ressource, welche die mit dem Ereignis assoziierte Aktion durchführt und damit das Ereignis auslöst, wohingegen das Objekt die System-Ressource ist, auf welches die Aktion angewendet wird. Die Aktion selbst ist neben dem Subjekt und Objekt der dritte Part, der die Beziehungsverbindung ausmacht und beschreibt die Operation, die das Subjekt auf das Objekt anwendet.

Die meisten Ereignisse der Prozess-Thread-Verhaltensebene können als eine Subjekt-Objekt-Relation modelliert werden. Es gibt auch Ereignisse für die dies nicht gilt (Inter-Modular-Transition, Task-Wechsel, Interrupts, hoch aggregierte Ereignisse der Meta-Ebene).

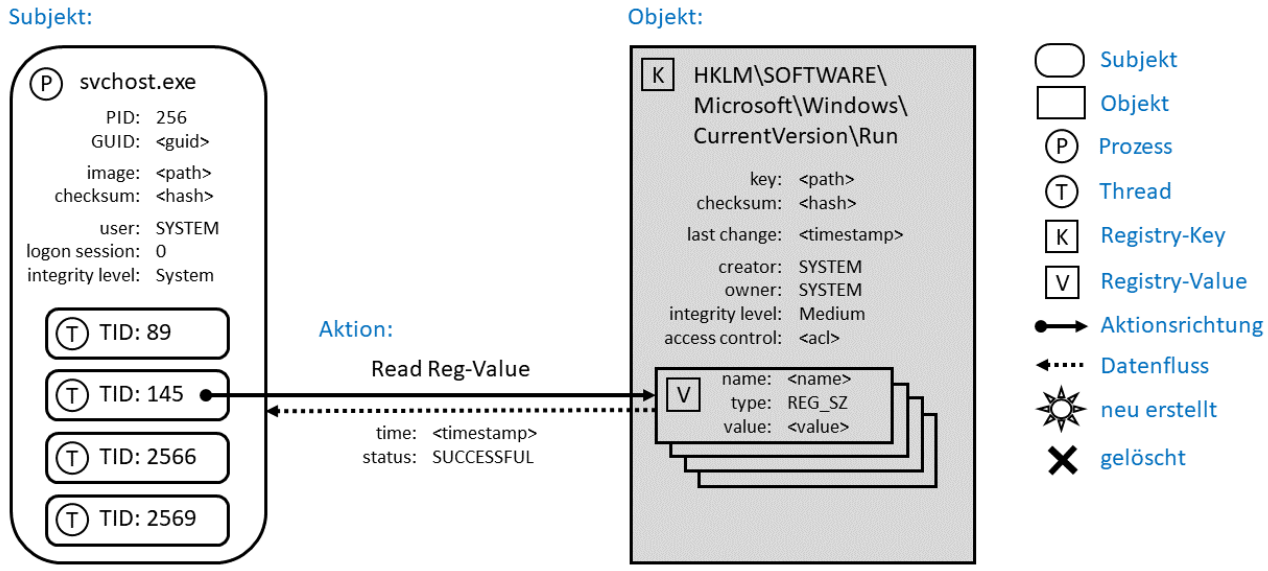

Abbildung 1.3: Subjekt-Objekt-Beziehung

1.4.1 Subjekt

Das Subjekt im Kontext von Prozess-/Threads-Verhaltensereignissen ist ein konkreter Thread des Systems, der den Code der Aktion durchführt, die das Ereignis auslöst.

Jeder Thread gehört eindeutig einem Prozess an. Da der Prozess-Kontext aus verschiedenen Gründen eine wichtige Rolle spielt, ist dieser als rahmende Struktur, die den Thread enthält, auch als Subjekt mit zu betrachten.

Im Falle aggregierter Ereignisse kann es vorkommen, dass mehrere Threads eines Prozesses als Subjekte an der Aktion beteiligt sind. Abhängig vom Grad der Abstraktion können auch mehrere Prozesse als Subjekte beteiligt sein.

1.4.1.1 Eigenschaften der Subjekte

Threads haben allgemein keine Bezeichnung oder Namen. Sie werden identifiziert über eine Nummer, die Thread-ID. Diese ist systemweit eindeutig für den Zeitraum, indem der Thread existiert. Ähnlich ist es auch bei Prozessen, auch diese besitzen eigentlich keinen Bezeichner bzw. Namen. Auch sie werden über eine für die Prozess-Lebensdauer systemweit eindeutige Nummer identifiziert, die Prozess-ID [MicCa][Yos20, S. 3ff]. Prozesse besitzen unter Umständen einen Bezeichner als zusätzliches Attribut, dieser ist jedoch nicht eindeutig und kann für die eindeutige Identifizierung und Adressierung eines einzelnen Prozesses nicht genutzt werden. Der Prozessbezeichner ist meist der Name des Executables, aus dem der Prozess instanziiert wurde, kann jedoch auch eine andere beliebige Zeichenkette sein, welche die Software definiert.

Da verwendete Prozess- und Thread-IDs nachdem Ende eines Prozesses oder Threads für einen neu erzeugten Prozess oder Thread erneut vergeben werden können, identifizieren sie das jeweilige Element nur für die Lebenszeit eindeutig. Für eine Ereignissprotokollierung ergibt sich daher das Problem, dass die gleiche Prozess- oder Thread-ID für eine längere Aufzeichnung von Ereignissen sich auf unterschiedliche Prozess- und Thread-Instanzen beziehen kann. Eine Lösung besteht darin, die IDs auf einen anderen Wert abzubilden, der über längere Zeiträume eine eindeutige Zuordnung liefert. Für dieses Ziel bieten sich GUIDs an. Das sind IDs, die konzeptionell so entwickelt sind und eine solche numerische Länge aufweisen, dass bei der korrekten Erstellung durch einen GUID-Algorithmus eine ID entsteht, die allumfassend eindeutig ist (zeitlich, räumlich, Kontext) oder zumindest das mehrfache Auftreten der ID verschwindend gering ist, sodass angenommen werden kann, dass eine erzeuge GUID weltweit kein zweites Mal erzeugt wurde und auch in Zukunft kein zweites Mal erzeugt werden wird [Wik22e]. Die Umsetzung von Prozess- und Thread-IDs auf GUIDs könnte so aussehen, dass bei jeder Erzeugung eines neuen Threads oder Prozesses eine neue GUID für dieses Element erzeugt wird. Diese GUID wird in einer Zuordnungstabelle mit der Prozess- oder Thread-ID des neu erzeugten Prozesses oder Threads verknüpft. In jedem von diesem Prozess oder Thread ausgelösten Ereignis, könnte in dem zugehörigen Event-Objekt die Attribute Thread-ID und Prozess-ID durch die jeweiligen GUIDs ersetzt oder ergänzt werden. Somit wäre in den im Event-Log gesammelten Event-Objekten auch über einen größeren Zeitraum hinweg eine eindeutige Zuordnung der Ereignisse zu den assoziierten Prozessen und Threads möglich. Die bereits erwähnte Ereignisbasierte Monitoring-Software Sysmon verwendet in den Attributen ihrer Event-Objekte zusätzlich zur Prozess-ID auch eine Prozess-GUID,

die Sysmon für jeden neuen Prozess erzeugt, intern in einer Zuordnungstabelle verwaltet und den gemessenen Ereignissen zuordnet.

Neben der ID/GUID über die Threads und Prozesse identifiziert werden, lassen sich für die Prozesse noch weitere wichtige Attribute benennen, die für die Rolle als Subjekt sehr relevant sind.

Ein Prozess wird im Normalfall aus einem Executable instanziiert, einer persistent gespeicherten Datei, die das Programm enthält. Ein Attribut, welches das Executables benennt, gibt somit Rückschluss auf das zugrundeliegende Programm des Prozesses. Um dieses mit anderen Executables (eventuell gleichen Namens) vergleichen und unterscheiden zu können, wäre ein Hash-Wert, der beim Start über die Daten des Executables gebildet wird, ein weiteres wünschenswertes Attribut.

Darüber hinaus kann beim Start eines Programms der erzeugende Prozess eine Liste an Start-Argumenten übergeben. Da das Verhalten des neu erzeugten Prozesses sich durch die Start-Argumente bestimmen kann, wären auch diese eine interessante Eigenschaft des Subjektes [YIRS17, S. 144ff][MicCa].

Prozesse besitzen zudem einen Arbeitskontext, der durch das Arbeitsverzeichnis (working directory) und einer Menge so genannter Umgebungsvariablen bestimmt wird. Die Umgebungsvariablen können beliebige Werte enthalten, die der Prozess als Konstanten nutzen kann. Sie sind standardmäßig durch die Anmeldesitzung definiert, können aber durch den erzeugenden Prozess für den neu erzeugten Prozess angepasst oder erweitert werden. Das Arbeitsverzeichnis bestimmt den Ordner im Dateisystem, der als Basisort für die Ausführung des Prozesses festgelegt ist. Benutzt der Prozess zum Adressieren von Dateien und Verzeichnissen relative Pfade, ist ihr relativer Bezugsort das spezifizierte Arbeitsverzeichnis. Auch das Arbeitsverzeichnis kann durch den erzeugenden Prozess modifiziert werden. Standardmäßig erbt der neu erzeugte Prozess das Arbeitsverzeichnis vom erzeugenden Prozess [Yos20, S. 3ff][MicCa].

Zudem wird jeder Prozess unter Windows im Kontext eines Benutzerkontos (User Account), einer Anmeldesitzung (Logon Session) und einer Terminal-Sitzung (Session) gestartet. Alle drei Attribute sind in den Verwaltungsdaten eines Prozesses über IDs spezifiziert [YIRS17, S. 32ff][YIRS17, S. 824ff]. Im Folgenden soll jedoch nur die Anmeldesitzung und das Benutzerkonto näher betrachtet werden.

Das Benutzerkonto in dessen Kontext ein Prozess ausgeführt wird, legt die Berechtigungen fest, die ein Prozess besitzt bezüglich des Zugriffs auf Systemressourcen. Der Betriebssystem-Kernel prüft bei jedem Zugriff auf Ressourcen, ob der Prozess über die notwendigen Berechtigungen verfügt, eine bestimmte Ressource in einer bestimmten Art und Weise zu nutzen [YIRS17, S. 716ff]. Technisch ist dies unter Windows über Access-Control-Listen (ACLs) umgesetzt, die für jedes vom System verwaltete Objekt

(Dateien, Verzeichnisse, Prozesse, Threads, IPC-Objekte, Geräte usw.) existieren. In ihnen ist für jedes Objekt eingetragen, welcher Benutzer welche Aktionen (lesen, schreiben, umbenennen usw.) auf das Objekt anwenden darf und welche nicht [YIRS17, S. 751ff]. Neben Benutzerkonten existieren noch Benutzergruppen, in denen mehrere Benutzerkonten oder Untergruppen zusammengefasst werden können. Statt Berechtigungen für einzelne Objekte für jedes einzelne Benutzerkonto zu spezifizieren, können diese auch für Benutzergruppen festgelegt werden. Die initialen Prozesse, die beim Start von Windows automatisch gestartet werden, werden im Kontext des Benutzerkontos „System“ ausgeführt, welches weitestgehend Vollzugriff auf alle Objekte besitzt [YIRS17, S. 773ff]. Erzeugt ein Prozess einen Kindprozess, wird dieser standardmäßig im gleichen Benutzerkontext wie der Elternprozess ausgeführt. Es ist dem Elternprozess jedoch auch möglich, den Kindprozess in einem anderen Benutzerkontext zu starten. Besitzt das Benutzerkonto des Elternprozess die Berechtigung, Prozesse im Kontext des gewählten anderen Benutzerkontos zu starten, kann dies ohne Interaktion durch den Benutzer durchgeführt werden. Dieser Vorgang lässt sich vielfach im System finden, bei dem ein Prozess eines höher privilegierten Benutzerkontos wie z.B. System einen Kind-Prozess im Kontext eines niedriger privilegierten Benutzerkontos wie z.B. dem Benutzerkontos eines Dienstes oder des realen Endanwenders startet. Möchte ein Prozess einen Kindprozess im Kontext eines Benutzerkontos starten, ohne die notwendigen Berechtigungen zu besitzen, ist dies unter Umständen auch möglich. In diesem Fall wird dem Endanwender durch das Betriebssystem eine Abfrage präsentiert, in der dieser bestätigen muss, dass der Prozess einen Kindprozess im Kontext des gewählten Benutzerkontos erzeugen darf mit eventueller Eingabe des Passworts dieses Benutzerkontos (UAC-Mechanismus) [YIRS17, S. 838ff]. Ein Prozess wird also bei seiner Erzeugung mit einem Benutzerkonto assoziiert, dass die Berechtigungen des Prozesses spezifiziert. Dies ist der initiale Benutzerkontokontext eines Prozesses. Unter Windows besitzen Prozesse jedoch zusätzlich die Möglichkeit, im Laufe ihres Lebens ihren eigenen Benutzerkontokontext zu wechseln. Für einen solchen Wechsel gelten dieselben Regeln, wie für die Erzeugung eines Kindprozesses unter einem anderen Konto. Nur mit den entsprechenden Berechtigungen kann ein Wechsel vollzogen werden. Erhöht ein Prozess seine Berechtigungen ist die Interaktion eines realen Endnutzers notwendig, der den Vorgang bestätigen muss. In den Kernel-internen Verwaltungsstrukturen besitzt ein Prozess dazu ein primäres Zugriffstoken oder Sicherheitstoken, das unter anderem den Benutzerkontokontext repräsentiert und bei der Prüfung der Zugriffsrechte Verwendung findet. Daneben können weitere Identitätswechseltokens existieren, die dann zum Einsatz kommen, wenn der Prozess zur Laufzeit seinen Benutzerkontokontext wechselt. Diese können statt dem gesamten Prozess auch einzelnen Threads zugeordnet werden. So können innerhalb eines Prozesses auch Threads existieren, die über andere Berechtigungen verfügen, als die regulären Threads des Prozesses.

Neben dem Benutzerkontokontext existiert auch noch der Anmeldesitzungskontext. Ein jeder Prozess unter Windows ist einer Anmeldesitzung zugeordnet. Der Anmeldesitzungskontext ähnelt sehr dem Benutzerkontokontext. Beide spezifizieren jedoch andere Aspekte des Prozesses. Während sich durch den Benutzerkontokontext die Berechtigungen des Prozesses ergeben, spezifiziert der Anmeldesitzungskontext den lokalen Pool an System-Objekten, auf dem der Prozess arbeitet, wenn dieser auf lokalen Objekten arbeitet. Konkret sind das die unter `HKCU` (Current User) bereitgestellten Registry-Schlüssel, die über `%user%` bereitgestellten Benutzerverzeichnisse und Anwendungskonfigurationen (AppData). Hinzu kommen UI-Ressourcen, die spezifisch für Anmeldesitzungen sind, wie ein gemeinsamer Desktop oder eine gemeinsame Zwischenablage (Clipboard), die sich alle Prozesse derselben Anmeldesitzung teilen [YIRS17, S. 824ff]. Der Anmeldesitzungskontext wird ähnlich wie beim Benutzerkontext bei der Erstellung standardmäßig vom Elternprozess geerbt. Prozesse mit speziellen Systemberechtigungen können einen Kindprozess in einer anderen Anmeldesitzung starten als der, der sie selbst zugeordnet sind. Dabei können sie komplett neue Anmeldesitzungen erstellen. Dies trifft z.B. für den Sitzungsmanagement-Prozess (`smss.exe`) des Windows-Systems zu. Der Benutzerkontokontext eines Prozesses und das assoziierte Benutzerkonto eines Anmeldesitzungskontextes können für einen Prozess verschieden ausfallen. So kann ein Prozess innerhalb der Anmeldesitzung eines speziellen Endanwender-Kontos mit Administratorberechtigungen ausgeführt werden. Der Benutzerkontokontext wäre in dem Fall das Administrator-Konto, welches die Berechtigungen bestimmt, der Anmeldesitzungskontext wäre jedoch unverändert die Sitzung des angemeldeten Endanwenders, auf dessen lokalen Daten und Einstellungen der Prozess arbeitet, wenn er lokale Objekte (Benutzerverzeichnisse, HKCU-Registry-Schlüssel) beim Kernel anfragt.

Die Anmeldesitzung (Logon Session) und das Benutzerkonto (User account) mit dem ein Prozess assoziiert ist bzw. in dessen Kontexten er ausgeführt wird, sind wichtige Attribute des Prozess-Subjektes, da sie wichtige Rahmenbedingungen beschreiben, die für das Verhalten des Prozesses gelten. Da Prozesse oder auch einzelne Threads wie beschrieben ihren Benutzerkontenkontext und somit ihre Berechtigungen anpassen können, könnte es interessant sein, das primäre Zugriffstoken und die Wechsel-Zugriffstoken für den Prozess und die Threads als separate Attribute des Subjektes zu führen. Dies ist abhängig vom Detailgrad der Systemüberwachung. Das primäre Sicherheitstoken oder der initiale Benutzerkontokontext des Prozesses sollten jedoch auf jeden Fall als Attribut des Subjektes protokolliert werden.

Initialer Prozessbaum unter Windows

Prozess	Kernel/User-Mode	Anmeldesitzung	Benutzeraccount
– Idle	Kernel-Mode	keine	System
– System	Kernel-Mode	0: System-Sitzung	System
—– smss.exe	User-Mode	0: System-Sitzung	System
——— smss.exe	User-Mode	0: System-Sitzung	System
———— csrss.exe	User-Mode	0: System-Sitzung	System
———— wininit.exe	User-Mode	0: System-Sitzung	System
————– services.exe	User-Mode	0: System-Sitzung	System
————— ...			
————– lsass.exe	User-Mode	0: System-Sitzung	System
————– fontdrvhost.exe	User-Mode	0: System-Sitzung	System
——— smss.exe	User-Mode	1: Benutzer-Sitzung 1	System
———— csrss.exe	User-Mode	1: Benutzer-Sitzung 1	System
———— winlogon.exe	User-Mode	1: Benutzer-Sitzung 1	System
————– userinit.exe	User-Mode	1: Benutzer-Sitzung 1	User ABC
————— Explorer.exe	User-Mode	1: Benutzer-Sitzung 1	User ABC
————— ...			
——— smss.exe	User-Mode	2: Benutzer-Sitzung 2	System
———— csrss.exe	User-Mode	2: Benutzer-Sitzung 2	System
———— winlogon.exe	User-Mode	2: Benutzer-Sitzung 2	System
————– userinit.exe	User-Mode	2: Benutzer-Sitzung 2	User XYZ
————— Explorer.exe	User-Mode	2: Benutzer-Sitzung 2	User XYZ
————— ...			

Tabelle 1.1: Initialer Prozessbaum unter Windows [AIRS21, S. 835]

Neben dem Zugriffskontrollsystem, das auf den Berechtigungen, die ein Prozess durch den Benutzerkontokontext besitzt, basiert, verwendet Windows seit Version Vista ein weiteres Zugriffskontrollsystem, das auf Basis von Integritätsleveln arbeitet. Beide Kontrollsysteme werden dabei parallel genutzt. Bei jedem Zugriff wird über beide Verfahren die Berechtigung geprüft [YIRS17, S. 720ff]. Dabei ergibt sich die effektive Berechtigung aus der stärksten Einschränkung, sodass ein Zugriff nur dann erlaubt ist, wenn beide Zugriffskontrollsysteme diesen erlauben. Unterbindet eines der beiden den Zugriff, wird dem Prozess die Aktion verweigert. Das neue Zugriffskontrollsystem auf Basis von Integritätsleveln wird als Mandatory-Integrity-Control (MIC) bezeichnet. Es ist kein Ersatz für das klassische Zugriffskontrollsystem sondern ergänzt dieses um den zusätzlichen Aspekt des Integritätslevels. Ab Windows Vista besitzen alle auf einem System laufenden Prozesse und alle vom System verwalteten Objekte (Dateien, Verzeichnisse, IPC-Objekte, Registry-Schlüssel, Services, Geräte usw.) ein zusätzliches Sicherheitsattribut, den Integritätslevel [YIRS17, S. 724f]. Der Integritätslevel soll die Vertrauenswürdigkeit eines Prozesses oder eines System-Objektes beschreiben. Windows sieht vier verschiedene Zustände (`Low`, `Medium`, `High`, `System`) vor, die der Integritätslevel einer jeden

Instanz annehmen kann. Bei einem Zugriff eines Prozesses auf ein Objekt wird der Integritätslevel des zugreifenden Prozesses mit dem Integritätslevel des Objektes, auf das zugegriffen wird, verglichen. Ist der Integritätslevel des Prozesses gleich oder größer dem Integritätslevel des Objektes, wird der Zugriff gewährt, andernfalls wird dieser verweigert. Prozesse, die im Kontext des Benutzerkontos „System“ ausgeführt werden, starten mit dem höchsten Integritätslevel `System`. Prozesse, die im Kontext des Benutzerkontos eines realen Endanwenders gestartet werden, bekommen beim Starten das Integritätslevel `Medium` zugewiesen [Wik21a][YIRS17, S. 724f]. Da durch von außen eingehende Inhalte Anwendungen exploitiert und für bösartige Zwecke missbraucht werden können, ist es ratsam, dass Anwendungen, die Inhalte von außerhalb des Rechners öffnen, wie beispielsweise Office-Anwendungen, PDF-Reader, E-Mail-Clients, Web-Browser oder andere Internet-Applikationen, mit dem Integritätslevel `Low` gestartet werden. Kommt es bei diesen Anwendungen zu einer Kontrollübernahme durch Angreifer, kann der Prozess aufgrund des niedrigen Integritätslevels nur noch stark eingeschränkt auf System-Objekte zugreifen. Diese Startoption muss entweder durch den Endanwender konfiguriert werden oder Anwendungen verringern selbstständig zur Laufzeit ihr Integritätslevel von `Medium` auf `Low`. Kindprozesse für die Browser-Tabs der Browser Microsoft Edge, Google Chrome und Mozilla Firefox nutzen diese Sicherheitsfunktion und verringern ihren Integritätslevel selbstständig [Wik21a]. Es ist dem Endbenutzer möglich Applikationen mit Administratorberechtigungen auszuführen. Beispielsweise benötigt Software zur Systemkonfiguration oder Installationsanwendungen diese Berechtigungen. Das Ausführen im Kontext des Benutzerkontos Administrator impliziert beim Prozess den Integritätslevel `High`. Unabhängig der Standardvergabe des Integritätslevels, kann dieser für einen neuen Kindprozess auch anders spezifiziert werden. Dabei kann ein Prozess den Integritätslevel jedoch gegenüber seinem eigenen nur beibehalten oder verringern. Das erhöhen des Integritätslevels eines Kindprozesses gegenüber des eigenen Levels bedarf spezieller Berechtigungen und der Interaktion des realen Endbenutzer (über User Account Control) [YIRS17, S. 838ff][Wik21a]. Objekte wie z.B. Dateien und Ordner, die vom Benutzer angelegt werden, bekommen initial den Integritätslevel `Medium` zugewiesen. Dateien des Betriebssystems besitzen meist den Integritätslevel `High` oder `System`. Der Integritätslevel eines Prozesses leitet sich ausdrücklich nicht aus dem Integritätslevel des Executables (EXE-Datei) ab. Der Integritätslevel für System-Ressourcen wie Dateien, Ordner, IPC-Objekte, Registry-Keys, Services, Geräte usw. kann von einem Prozess verändert werden, wenn dieser die Berechtigungen besitzt und sein Integritätslevel höher ist, als das des Objektes, dessen Integritätslevel verändert werden soll. So kann z.B. der Endanwender in Form des Administrators die Integritätslevel der Dateien, Verzeichnisse und Registry-Schlüssel konfigurieren. Unter den meisten Varianten des Windows-Betriebssystems ist dazu eine spezielle Software notwendig. Der Integritätslevel wird in der Regel nicht in den Objekt-Eigenschaftsfenstern des Windows Explorers oder des Registry-Editors angezeigt [YIRS17, S. 724f].

Der Integritätslevel ist somit neben dem Benutzerkontokontext und dem Anmeldesitzungskontext ein weiteres sehr wichtiges Attribut des Subjektes Prozess, da es ebenso starke Rahmenbedingungen für das Verhalten eines Prozesses charakterisiert.

Aufbau einer Beispiel-SID

S-1-5-21-7623811015-3361044348-030300820-1013

Erläuterung:

Kurzzeichen für SID:	S
Revisionsnummer der SID-Version:	1
'Identifier Authority' (Ersteller der SID):	5
ID der Domänen oder des lokalen Systems:	21-7623811015-3361044348-030300820
Benutzernummer / Gruppennummer:	1013 (Relative ID, RID, die bei lokalen Endbenutzer-Accounts mit 1000 beginnend hochgezählt wird)

Erlaubte Werte von 'Identifier Authority':

0	Null-account Authority
1	World Authority
2	Local Authority
3	Creator Authority
4	Non-unique Authority
5	NT Authority
9	Resource Manager Authority
16	Mandatory Integrity Level

SIDs für Integritätslevel:

Low	S-1-16-4096
Medium	S-1-16-8192
High	S-1-16-12288
System	S-1-16-16384

Tabelle 1.2: Aufbau einer Beispiel-SID [Wik20b]

Unter Windows werden Benutzerkonten, Benutzergruppen und Integritätslevel über SIDs (Security Identifier) eindeutig identifiziert. SIDs sind den zuvor beschriebenen GUIDs vom Konzept her ähnlich, jedoch handelt es sich um eine Microsoft-eigene Variante, einen global eindeutigen Identifier für sicherheitsrelevante System-Objekte zu implementieren. In Windows sollen für die Identifikation von Benutzerkonten, Gruppen und Integritätslevel ausdrücklich nicht die Namen benutzt werden, da diese unter Umständen nachträglich geändert werden können. Stattdessen soll eine ID benutzt werden, die so gestaltet ist, dass sie dauerhaft beibehalten werden kann. Zudem sollen Benutzer, Gruppen, Integritätslevel gleichen Namens zwischen verschiedenen Rechnern unterschieden werden können, da diese ja nicht dieselbe Instanz darstellen. Dies ist wichtig für die Windows-Sicherheitsfeatures im Kontext von Windows-Netzwerkprotokollen mit Ordnerfreigaben und verteilten Benutzerkonten mittels Active-Directory und Domänencontroller-Infra-

struktur. Die Prozess-Attribute Benutzerkontokontext und Integritätslevel in den Verwaltungsstrukturen eines Prozesses verwenden SIDs um die entsprechenden Objekte zu identifizieren [YIRS17, S. 720ff][Wik22d][Wik20b].

1.4.2 Objekt

Die Objekte im Kontext der Prozess/Thread-Verhaltensanalyse sind Systemressourcen, die vom Betriebssystem verwaltet werden und durch System-Objekte abstrahiert werden, auf denen die Subjekte also die Prozesse und Threads zugreifen. Dies können z.B. Dateien, Verzeichnisse, Registry-Schlüssel, IPC-Objekte, wie Pipes, Message-Queues, Mutexe, Semaphoren, Netzwerk-Sockets, Dienste, Geräte, Benutzerkonten, Benutzergruppen (Aufzählung ist nicht vollständig) und auch andere Prozesse und Threads sein [AIRS21, S. 125ff]. Die letztgenannten Prozesse und Threads können also sowohl die Rolle des Subjektes als auch die Rolle des Objektes in einer durch ein Ereignis beschriebenen Beziehung einnehmen.

1.4.2.1 Eigenschaften der Objekte

Objekte können über ein Attribut oder eine Kombination von Attributen eindeutig auf dem System identifiziert werden. Je nach Objekttyp kann dieses eindeutig identifizierende Attribut ganz unterschiedlich aussehen. Manche Objekte lassen sich über IDs oder spezielle SIDs (Security Identifier) adressieren. Andere Objekte werden über einen Bezeichner und einen Pfand eindeutig adressiert (Registry-Keys, Dateien, Verzeichnisse, IPC-Objekte, Geräte). Es existieren für die unterschiedlichen Typen von Objekten verschiedene Namensräume. Dateien und Ordner auf Datenträgern befinden sich in den Namensräumen der Laufwerke, die über den Dateimanager des Betriebssystem-Kernels bereitgestellt werden. Registry-Keys befinden sich im Namensraum der Registrierungsdatenbank, der vom Config-Manager des Kernels bereitgestellt wird. Die meisten anderen System-Objekte für Interprozesskommunikation, Synchronisation, Geräte usw. befinden sich im Namensraum für NT-System-Objekte, der vom Object-Manager des Kernels verwaltet wird [AIRS21, S. 125ff]. Neben dem Verzeichnissystem des Objekt-Manangers bestehen noch weitere Namensräume wie z.B. PipeFS für Pipes oder der WinSock-Namespace mit den Sockets.

Manche Objekte kapseln einen inneren Zustand. Bei Registry-Schlüsseln wären das z.B. die Werte der Felder eines Schlüssels und eventuell der innere Zustand aller Kind-Schlüssel. Bei Dateien ergibt sich der innere Zustand durch den Inhalt der Datei. Ein solcher innerer Zustand von Objekten könnte über einen Hash-Wert charakterisiert werden. Zum Beispiel könnte ein Hash-Wert über den Inhalt einer Datei gebildet werden.

Dies ermöglicht in einer einfachen Weise, an einem numerischen Wert zu erkennen, ob z.B. verschiedene Dateien den gleichen Inhalt haben oder ob sich der Inhalt derselben Datei verändert hat, wenn der Hash-Wert sich geändert hat.

Für Objekte die Speicher belegen können meist eine oder mehrere Größen spezifiziert werden. Dies ist intuitiv meist nur für die Objekte einfach möglich, die direkt Speicherplatz auf einem Datenträger oder in einem Puffer belegen wie z.B. Dateien. Bei Dateien können Größenangaben für den Inhalt und für die gesamte Datei einschließlich der Verwaltungsstrukturen gefunden werden.

Werden Objekte neu erstellt, geschieht dies durch einen Prozess, der in einem Benutzerkontokontext ausgeführt wird. Das Benutzerkonto dieses Kontextes ist der Ersteller (Creator) des Objektes. Der Ersteller ist initial auch der Besitzer (Owner) des Objektes. Das Besitzverhältnis kann jedoch im Nachhinein auch zu einem anderen Benutzerkonto geändert werden. Solche Änderungen an den Sicherheitsattributen eines Objektes können wiederum von Prozessen durchgeführt werden, wenn diese bezüglich des Objektes die notwendigen Berechtigungen besitzen [YIRS17, S. 751ff].

Fast alle Objekte, die vom Windows-Betriebssystem verwaltet werden, besitzen pro Objekt eine Access-Control-Liste (ACL), die den Zugriff von Prozessen auf dieses Objekt regelt. Die Access-Control-Liste (ACL) eines Objektes enthält Einträge, die spezifizieren, welcher Benutzer oder welche Gruppe welche Aktionen (lesen, schreiben, verwenden, Attribute ändern usw.) durchführen darf. Der Besitzer eines Objektes genießt meist besondere Privilegien bezüglich der Zugriffsrechte auf das Objekt das er besitzt. Diese Berechtigungen werden unter Windows jedoch wie alle anderen als Eintrag über die Access-Control-Liste (ACL) des Objektes realisiert. Speziell ist jedoch das dort der Meta-Benutzer „Besitzer“ oder „owner“ eingetragen werden kann, der dem Benutzeraccount entspricht, der das Objekt besitzt. Initial besitzt der Besitzer eines Objektes in der Regel einen Vollzugriff auf das Objekt [YIRS17, S. 751ff].

Wie Prozesse besitzen auch die Objekte seit Windows Vista das zusätzliche Sicherheitsattribut „Integrity Level“. Das Integritätslevel eines Objektes entspricht initial dem Integritätslevel des Prozesses, der das Objekt anlegt (ausgenommen Kindprozesse). Der Integritätslevel ist Bestandteil des mit Windows Vista neu eingeführten Zugriffsschutzsystem Mandatory-Integrity-Control (MIC). Dabei wird bei einem Zugriff eines Prozesses auf ein Objekt die Integritätslevel beider Instanzen miteinander verglichen. Damit das Betriebssystem den Zugriff erlaubt, muss das der Integritätswert des Objektes kleiner oder gleich dem des zugreifenden Prozesses sein. Das MIC-Verfahren wird zusätzlich zum klassischen Zugriffsschutz via Access-Control-Listen (ACLs) eingesetzt, wobei für das Erteilen des Zugriffs beide Verfahren den Zugriff erlauben müssen [Wik21a].

Technisch besitzen die Windows-System-Objekte intern Security-Deskriptoren, welche die beschriebenen Sicherheitsmerkmale wie Besitzverhältnis, Access-Control-Listen (ACLs), Integritätslevel usw. beinhalten. Für jedes Objekt besteht in dessen Verwaltungsstrukturen ein Security-Deskriptor. Bei Objekten, die persistent gespeichert werden, wird der Security-Deskriptor als Teil der Verwaltungsstruktur binär serialisiert auf dem Datenträger abgelegt. Bei Dateien befindet er sich in der NFTS-Verwaltungsstruktur der Dateien auf dem Datenträger. Bei manchen Verwaltungsobjekten wird er z.B. in Binär-Feldern von Registry-Schlüsseln persistiert (ETW-Objekte).

Zuletzt sind noch zeitliche Attribute für ein Objekt auszumachen. So stellt der Erstellungszeitpunkt eine interessante Eigenschaft dar. Daneben kann noch der Zeitpunkt des letzten Zugriffs, der letzten Verwendung eines Objektes interessant sein. Bezüglich Objekten, die einen änderbaren internen Zustand besitzen, wäre der Zeitpunkt der letzte Änderung eine wichtige Charakteristik des Objektes [Mic4d].

1.4.3 Aktion

Subjekt und Objekt werden durch eine Aktion in Beziehung gebracht, die die Operation beschreibt, die das Subjekt auf das Objekt anwendet. Die Aktion ist dabei eine eindeutige Operation aus einer endlichen Menge bekannter Operationen.

1.4.3.1 Eigenschaften der Aktion

Die Operation der Aktion wird durch eine eindeutige Operationsbezeichnung definiert, die der Aktion die Operation aus der bekannten endlichen Menge an möglichen Operationen zuordnet. Je nach Ereignis-Verfolgungssystem kann die Operation der Aktion aus dem Ereignistyp des Event-Objektes abgeleitet werden. Dieser könnte sich z.B. aus einer Event-ID, einem Op-Code oder einer Kombination von Event-ID und Ereignisquelle (Provider-ID) zusammensetzen [Mic5e].

Für eine Aktion können verschiedene Attribute bestimmt werden. Die wichtigsten sind der Zeitpunkt, wann die Aktion stattgefunden hat und der Status, ob die Aktion erfolgreich war oder gescheitert ist. Der Zeitpunkt entspricht dabei dem Auftrittszeitpunkt des assoziierten Ereignis.

Zudem weisen manche Aktionen einen Datenfluss auf. Z.B. fließen Daten beim Schreiben in eine Datei vom Adressraum des Prozesses also vom Subjekt in die Datei also in das Objekt. Im Gegenzug findet z.B. beim Lesen eines Wertes aus einem Registry-Schlüssels ein Datenfluss vom Registry-Key-Objekt in den Adressraum des Prozesses statt. Nicht jede Aktion besitzt einen Datenfluss von Nutzdaten zwischen Objekt und

Subjekt. Beispiele wären unter anderem das Löschen eines Objektes oder das Erzeugen eines Prozesses oder Threads. Für Aktionen, bei denen ein Datenfluss von Nutzdaten stattfindet, ist die Richtung des Datenflusses ein weiteres wichtiges Attribut der Aktion. Über solche Richtungsinformationen können beispielsweise in der weiteren Analyse von Ereignissen der Abfluss von geschützten Informationen oder das Kompromittieren bestimmter Komponenten über mehrere Ereignisse hinweg verfolgt werden.

1.5 Ereignisse

Im folgenden Abschnitt werden verschiedene Ereignisse detaillierter vorgestellt, die für die Verhaltens-Überwachung von Prozessen interessant sind. Dabei wird besonders auf den Anwendungsfall der Angriffserkennung und Malwareanalyse auf Basis von Prozessverhaltensereignissen eingegangen. Bezüglich der Abstraktionsebene, sind im Folgenden nur Ereignisse der Thread/Prozessverhaltensebene beschrieben, da diese Abstraktionsebene für die in dieser Arbeit betrachteten Aspekte am relevantesten erscheint.

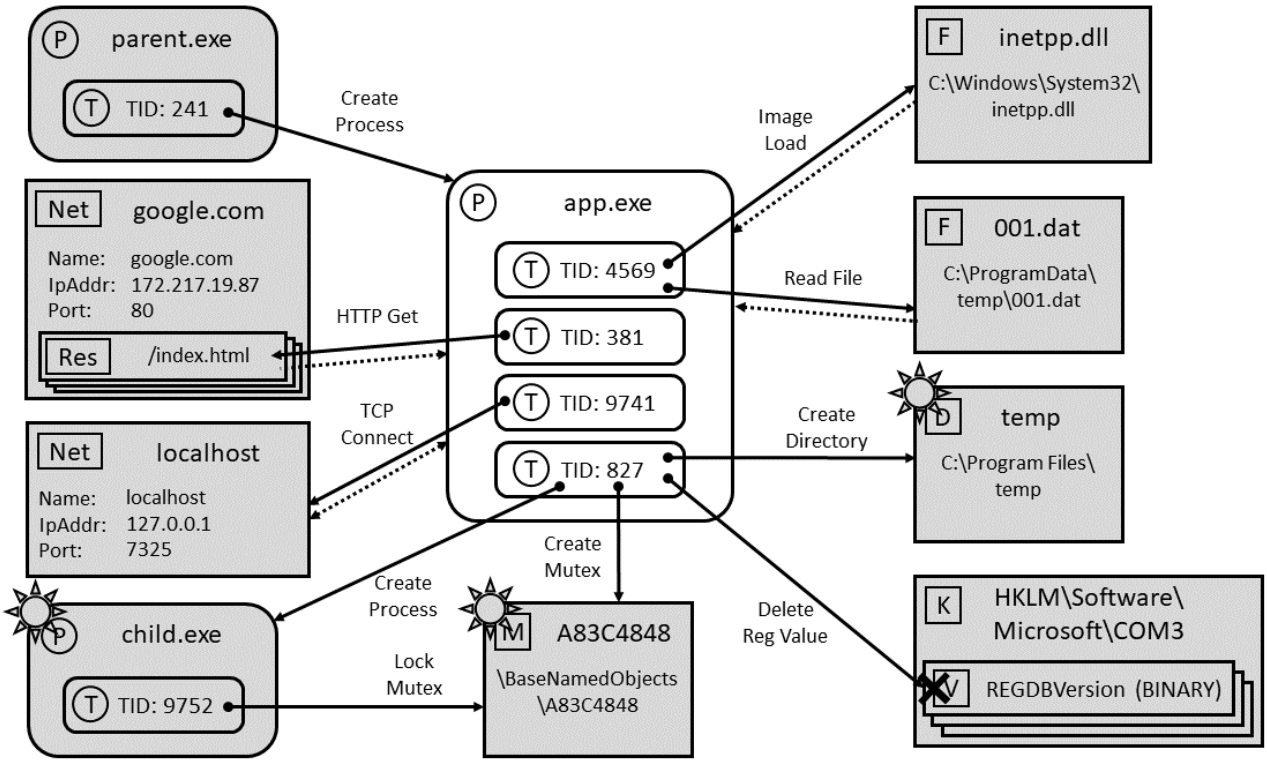

Abbildung 1.4: Ereignistypen - Überblick

1.5.1 Prozess-Verwaltung

Wird im Betrieb eines Computersystems ein weiteres Programm gestartet, wird ein neuer Prozess instanziiert. Die Entstehung neuer Prozesse geschieht jedoch nicht kontextlos sondern ist immer die Folge einer Prozesserzeugungsaufforderung durch einen bereits vorhandenen Prozess. Jeder Prozess ist somit von einem vorher vorhandenen Prozess erzeugt worden, welcher der Elternprozess (parent process) des neu erzeugten Prozesses ist. Nach dem Booten des Betriebssystems existiert ein initialer Prozess, der keinen Elternprozess besitzt. Dieser Prozess erzeugt weitere Prozesse als seine Kindprozesse (child process), welche wiederum weitere Prozesse erzeugen. Das Prozedere lässt sich in gleicher Weise fort denken. Somit sind zumindest initial alle Prozesse indirekt miteinander verbunden. Sie bilden einen Baum aus Eltern-Kind-Beziehungen. Im Laufe der Zeit bricht dieser Baum an machen Stellen jedoch in Teilbäume auseinander. Dies geschieht durch das Beenden einzelner Prozesse, welche Kind-Prozesse besitzen. Es gibt je nach Implementierung eines Programms verschiedene Möglichkeiten, hinsichtlich der Weiterexistenz von Kindprozessen, wenn der Elternprozess beendet wird. Zum einen ist es möglich, dass der Elternprozess seine Kindprozesse beendet oder ihnen mitteilt, dass sie sich selbst beenden sollen. Alternativ kann der sich beendende Elternprozess seine Kindprozesse weiterlaufen lassen. In diesem Fall verlieren die Kindprozesse ihren Elternprozess und werden somit vom Baum getrennt. Sie bilden abgetrennte Einzelelemente oder, wenn sie selbst Kindprozesse besitzen, abgetrennte Teilbäume aus. Unter Linux werden solche abgetrennten Strukturen dem initialen Prozess „init“ zugeordnet und ihre Elternprozess-ID dementsprechend mit der Prozess-ID vom „init“-Prozess belegt. Unter Windows verbleiben sie auf oberster Ebene abgetrennt vom restlichen Prozessbaum. Die Elternprozess-ID wird nicht aktualisiert [MicCb][MicCa].

Ein Prozess kann aus verschiedenen Gründen beendet werden. Z.B. kann das Ende des Programms erreicht werden und der Prozess beendet sich selbst. Selbiges passiert auch im Falle eines Fehlers, der vom Programm mit einer Beendigung behandelt wird. Alternativ können auch andere Prozesse einen Prozess beenden, wenn diese über entsprechende Berechtigungen verfügen. In der Regel besitzt ein Elternprozess die Berechtigung seine eigenen Kindprozesse und diesen untergeordnete Prozesse zu beenden. Berechtigungen werden über, mit der Ressource verknüpften, Security-Deskriptoren mit ACLs geregelt. Die effektiven Berechtigungen sind vom Benutzerkontokontext abhängig, in dem ein Prozess ausgeführt wird. Prozesse, die als „Administrator“ (Windows) ausgeführt werden, sind in der Regel berechtigt andere Prozesse zu beenden. Darüber hinaus ist auch der Kernel-Code des Betriebssystems allgemein in Lage, jeden Prozess zu beenden. Dies ist beispielsweise notwendig, wenn Prozesse in ihrer Ausführung hängen geblieben sind und somit nicht mehr in der Lage sind, sich selbst zu beenden.

Das Erzeugen und Beenden von Prozessen stellt somit eine Gruppe elementarer Aktionen dar, die für die Überwachung des Prozess- und Systemverhaltens sehr relevant ist. Als generische Event-Objekte für eine Ereignisprotokollierung können folgende Ereignisse spezifiziert werden:

process create	Kindprozess erzeugt	processID/GUID, threadID/GUID, time, createdProcessID/GUID, ExecutalePath, ExecutableChecksum, User, LognonSession, IntegrityLevel
process delete	anderen Prozess beendet	processID/GUID, threadID/GUID, time, deletedProcessID/GUID
process exit	sich selbst beendet	processID/GUID, threadID/GUID, time

Tabelle 1.3: Generische Prozess-Ereignistypen

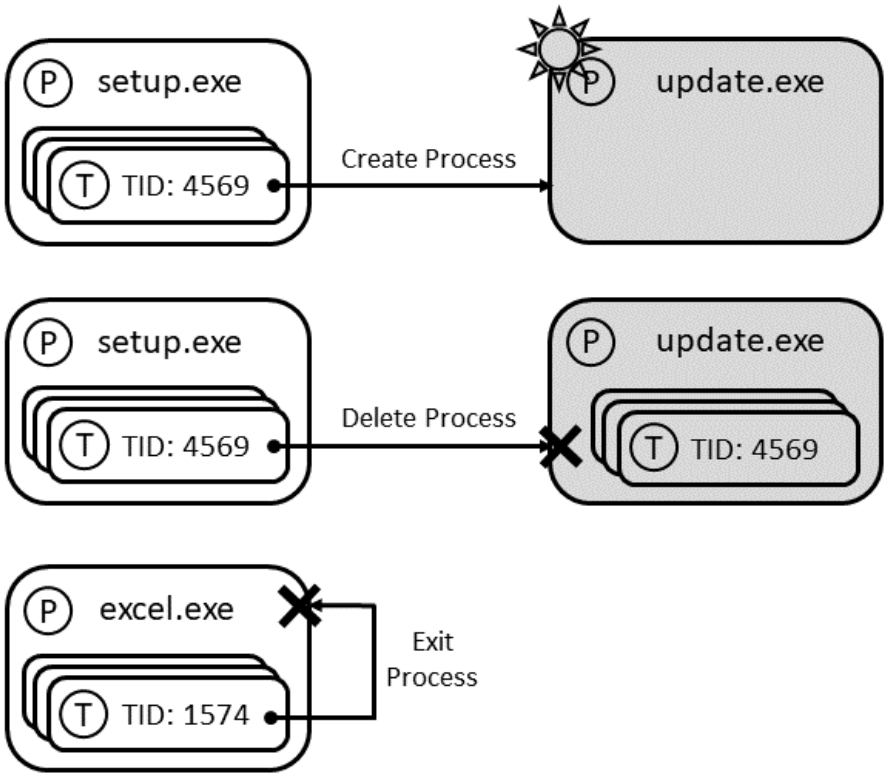

Abbildung 1.5: Generische Prozess-Ereignistypen

1.5.2 Thread-Verwaltung

Zu Beginn eines Prozesses existiert ein Thread, der als Main-Thread bezeichnet wird. Dieser wird unmittelbar durch die Prozesserzeugung mit angelegt [YIRS17, S. 144ff]. Ähnlich wie bei den Prozessen, fangen weitere Threads in der Regel dadurch an zu existieren, indem bereits vorhandene Threads diese erzeugen. Somit gehen meist alle Threads eines Prozesses direkt oder indirekt aus dem Main-Thread bzw. dessen Kind-Threads hervor. Ein Thread kann sich selbst beenden, indem er zum Ende seiner Ausführung kommt, oder das Betriebssystem anweist, sich zu beenden. Darüber hinaus kann ein Thread auch von anderen Threads des selben Prozesses beendet werden. Neben dem Beenden können Threads auch gestartet und gestoppt werden. Das Starten und Stoppen eines Threads bezieht sich auf den Ausführungszustand des Threads. Ein pausierter Thread steht nicht bereit, um auf einem CPU-Kern ausgeführt zu werden. Dahingegen kann ein ausführbarer Thread auf dem CPU-Kern ausgeführt werden, wenn er sich im Zustand „bereit“ befindet. Der Code, welcher innerhalb eines Prozesses ausgeführt wird, kann mehr oder weniger durch den Entwickler des Programms kontrolliert werden und ist dafür aus Sicht des Programm-Entwicklers vertrauenswürdig (angenommen, dass dem Betriebssystem-Code und dem Code von verwendeten Bibliotheken generell vertraut werden kann). Daher bedarf es keiner Prüfung von Berechtigungen bezüglich der Thread-Steuerung durch andere Threads des selben Prozesses [YIRS17, S. 20ff][MicCa].

Windows sieht jedoch Möglichkeiten vor, dass auch Threads in anderen Prozessen erstellt und gesteuert werden können. Solche Threads in fremden Prozessen werden als Remote-Threads bezeichnet [Hos17]. Diese Funktionalität ist sicherheitskritisch, da hierdurch ein fremder Prozess von außen manipuliert werden kann. Sie sollte daher durch ein Berechtigungsmanagement kontrolliert werden, sodass nur Prozesse mit besonderen Administratorrechten solche Dinge durchführen können. Da letzteres aus Gründen der Abwärtskompatibilität in der Vergangenheit unzureichend unter Windows umgesetzt wurde, ist dieser Mechanismus von Schadsoftware vielfach ausgenutzt worden. Die bösartige Methode wird als Thread-Injection bezeichnet, bei der ein bösartiger Prozess bösartigen Code in den Adressraum eines fremden Prozesses einbringt, in diesem einen Remote-Thread erzeugt, die Startadresse des Threads so konfiguriert, dass diese auf den bösartigen Code zeigt und diesen dann im fremden Prozess zur Ausführung bringt [Hos17]. Dadurch wird ein legitimer Prozess infiziert und entführt. Er wird somit zu einem bösartigen Prozess, der schadhafte Vorgänge durchführen kann.

Das Erzeugen und Beenden von Threads stellt somit einen bedeutsamen Aspekt des prozessinternen Verhaltens dar. Als generische Event-Objekte für eine Ereignisprotokollierung können folgende Ereignisse genannt werden:

thread create	neuen Thread erstellt	processID/GUID, threadID/GUID, time, createdThreadID/GUID, startAddr, stackBaseAddr
thread delete	anderen Thread beendet	processID/GUID, threadID/GUID, time, deletedThreadID/GUID
thread start	Thread gestartet	processID/GUID, threadID/GUID, time, startedThreadID/GUID
thread stop	Thread gestoppt	processID/GUID, threadID/GUID, time, stoppedThreadID/GUID
thread exit	hat sich selbst beendet	processID/GUID, threadID/GUID, time

Tabelle 1.4: Generische Thread-Ereignistypen

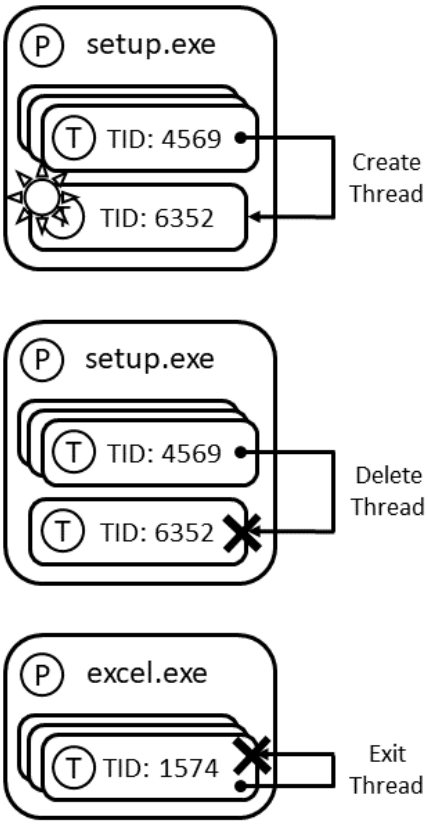

Abbildung 1.6: Generische Thread-Ereignistypen

1.5.3 Abbild-Ladevorgänge

Programme verwenden häufig Bibliotheken oder andere Ressourcen, die zur Laufzeit in den virtuellen Adressraum eines Prozesses eingeblendet werden müssen. Der Betriebssystem-Kernel stellt dem Prozess Funktionalitäten zur Verfügung, solche Bibliotheken oder andere Ressourcen in den Speicher einzublenden und gegebenenfalls dynamisch zu linken.

Dynamisch gelinkte Bibliotheken und andere ladbare Ressourcen, welche bereits in den PE-Lademodulen (EXE, DLL, OCX, CPL, SYS, DRV) über die Import-Tabellen spezifiziert sind, werden bei der Prozessinstanziierung als Image-Abbild vom Betriebssystem-Kernel im virtuellen Prozessadressraum zur Verfügung gestellt [YIRS17, S. 144ff]. Darüber hinaus ist es dem Prozess möglich, weitere Bibliotheken oder andere ladbare Ressourcen zu einem späteren Zeitpunkt der Ausführung nachzuladen oder wieder zu entladen. Dazu stellt die Windows-API entsprechende Funktionen zur Verfügung. Die nachgeladenen Module müssen dabei nicht im Vorhinein bekannt sein. Ihr Name, Pfad sowie die Intention zum Nachladen selbst können dynamisch als Ergebnis der Ausführung einer Anwendung entstehen. Sie stehen also eventuell erst zur Laufzeit fest [YIRS17, S. 173ff].

Im Malware-Kontext wäre z.B. der Fall von Interesse, bei dem ein Prozess eine potentiell bösartige Bibliothek lädt. In diesem Fall wird potentiell bösartiger Code in den Adressraum des Prozesses eingebracht und kommt dort eventuell zur Ausführung. Diese kompromittiert den Prozess und macht ihn potentiell bösartig.

Es resultieren die zwei generischen Lade-Ereignisse „image load“ und „image unload“:

image load	Image geladen	ProcessID/GUID, ThreadID/GUID, Time ImageName, ImagePath, ImageChecksum, ImageBaseAddr
image unload	Image entladen	ProcessID/GUID, ThreadID/GUID, Time ImageName

Tabelle 1.5: Generische Image-Load-Ereignistypen

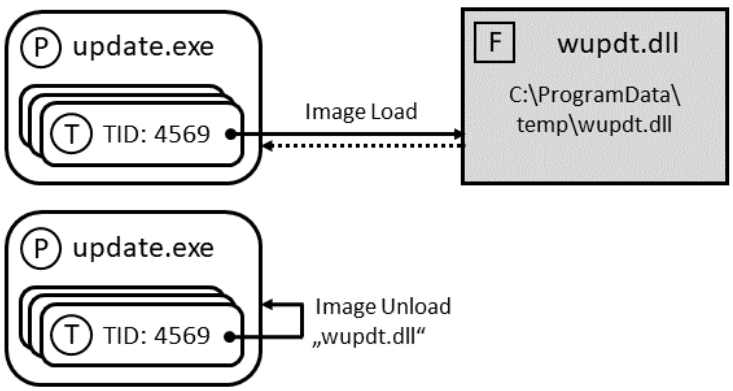

Abbildung 1.7: Generische Image-Load-Ereignistypen

1.5.4 Datei-Zugriffe

Dateien und Verzeichnisse auf Datenträgern sind die elementaren Organisationsstrukturen, um Daten persistent zu speichern. Der Betriebssystem-Kernel bietet Prozessen die Funktionalität, Daten in Container-Strukturen wie Dateien abzulegen und diese in hierarchischen Verzeichnisstrukturen (Bäume) zu verwalten. Dabei sind Verzeichnisse und Dateien über Bezeichner (Dateinamen) und Bezeichner-Ketten (Pfade) eindeutig adressierbar. Die Daten der Dateien und Verzeichnisstrukturen werden dazu systemintern in Dateisystemen organisiert und in Partition von Datenträgern bzw. Massenspeichergeräten (IO-Geräte) persistent gespeichert. Die technische Organisation wird von Dateisystem-Treibern innerhalb des Betriebssystem-Kernels durchgeführt und dadurch für den Anwendungs-Code abstrahiert. Der Prozess arbeitet auf abstrakten Datei- und Verzeichnis-Objekten, die ihm als Systemressource vom Betriebssystem-Kernel zur Verfügung gestellt werden.

Um auf Dateien oder Verzeichnissen zu arbeiten, also aus ihnen zu lesen, in sie zu schreiben, ihren Elemente abzufragen, zu ergänzen oder zu ändern, müssen Dateien und Verzeichnisse geöffnet werden. Dabei stellt der Betriebssystem-Kernel dem Prozess die Ressource bereit und liefert dem Prozess eine „Handle“ genannte Referenz auf das Datei- oder Verzeichnis-Objekt. Durch weitere Vorgänge, in denen der Prozess den Handle benutzt, um die geöffnete Ressource zu referenzieren, kann auf den Objekten gearbeitet werden. Es können Daten in die Datei geschrieben werden, aus ihr gelesen werden. Dateien können zudem angelegt, gelöscht, umbenannt und ihre Attribute und Zugriffsrechte abgefragt und bearbeitet werden. Der Betriebssystem-Kernel prüft beim durchführen der Aktionen, insbesondere beim Öffnen der Objekte mit bestimmten Zugriffsrechten (z.B. Öffnen zum Lesen, Öffnen zum Schreiben, Öffnen zum Lesen und Schreiben), ob der Prozess berechtigt ist, die entsprechenden Zugriffe auf dem Ziel-Objekt durchzuführen. Der Zugriff auf Dateien und Verzeichnisse kann sich also wie beschrieben aus mehreren Einzelereignissen, wie z.B. Öffnen, Schreiben, Schließen zusammensetzen. Solche Einzelereignisse lassen sich eventuell aber auch zu einem aggregierten Datei-Zugriffsereignis zusammenfassen [YIRS17, S. 33ff][Yos20, S. 15ff][AIRS21, S. 125ff].

Da die persistente Speicherung von Daten auf Massenspeichern und das Auslesen dieser ein sehr elementares Verhalten von Prozessen darstellt, ist die Protokollierung von Ereignissen, die Zugriffe auf Dateien anzeigen, ein elementarer Aspekt der Verhaltensüberwachung von einzelnen Prozessen oder dem gesamten System. Auch im Kontext von Angriffen und der Verhaltensanalyse von Schadsoftware spielt der Zugriff auf Dateien eine zentrale Rolle. Angriffe können das Ziel haben Informationen auszuleiten oder Daten auf einem angegriffenen System zu manipulieren. Diese Daten und Informationen liegen meist in Dateien auf den angegriffenen Systemen vor, auf die die Malware oder der Angreifer zugreifen muss. Schadsoftware legt sich im Rahmen der Infektionskette

persistent in Dateien auf dem System ab, um auch nach einem Neustart des infizierten Systems wieder aktiv werden zu können. Daher ist die ereignisbasierte Überwachung von Datei-Zugriffen auch im Kontext der Angriffserkennung sehr relevant.

Für die Protokollierung von Datei- und Verzeichniszugriffen können unter anderem folgende generische Ereignisse genannt werden:

file/dir create	Datei oder Verzeichnis erstellt	ProcessID/GUID, ThreadID/GUID, Time, FileHandle, FilePath
file/dir delete	Datei oder Verzeichnis gelöscht	ProcessID/GUID, ThreadID/GUID, Time, FilePath
file/dir rename	Datei oder Verzeichnis umbenannt	ProcessID/GUID, ThreadID/GUID, Time, FilePath, NewName
file/dir open	Datei oder Verzeichnis geöffnet	ProcessID/GUID, ThreadID/GUID, Time, FileHandle, FilePath, AccessMode(R/W), FileSize, FileChecksum
file/dir close	Datei oder Verzeichnis geschlossen	ProcessID/GUID, ThreadID/GUID, Time, FileHandle, FileSize, FileChecksum
file read	Daten aus Datei gelesen	ProcessID/GUID, ThreadID/GUID, Time, FileHandle, Offset, ChunkSize
file write	Daten in Datei geschrieben	ProcessID/GUID, ThreadID/GUID, Time, FileHandle, Offset, ChunkSize, FileSize, FileChecksum
file/dir attribute read	Attribut gelesen	ProcessID/GUID, ThreadID/GUID, Time, FileHandle, AttributeIndex, ReadValue
file/dir attribute write	Attribut geändert	ProcessID/GUID, ThreadID/GUID, Time, FileHandle, AttributeIndex, WriteValue
dir enum	Verzeichnisinhalt abgefragt	ProcessID/GUID, ThreadID/GUID, Time, FileHandle
file mapping create	Datei in Adressraum eingeblendet	ProcessID/GUID, ThreadID/GUID, Time, FilePath, AccessMode(R/W) FileSize, FileChecksum, ImageBaseAddr

Tabelle 1.6: Generische Datei-Ereignistypen

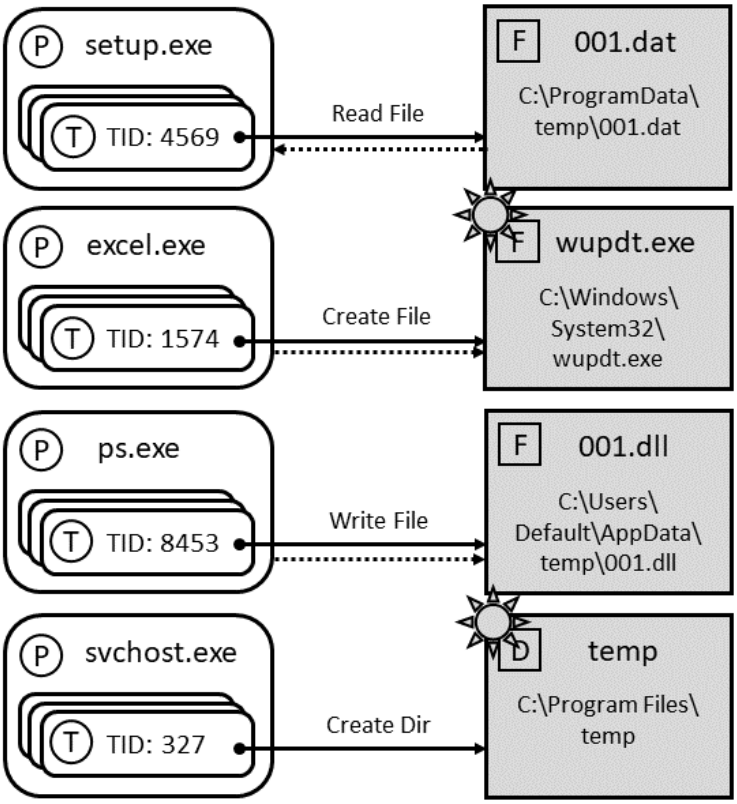

Abbildung 1.8: Generische Datei-Ereignistypen

1.5.5 Registry-Zugriffe

Der Kernel des Windows-Betriebssystems stellt Anwendungen und Diensten eine zentrale Konfigurationsdatenbank bereit, die Windows-Registrierungsdatenbank (engl. Windows-Registry) genannt wird. In dieser liegen die Konfigurationsparameter des Betriebssystems sowie der installierten Anwendungen und Dienste. Während unter UNIX-basierten Betriebssystemen die Software ihre Konfiguration vielen einzelnen textbasierten Konfigurationsdateien entnimmt, sieht Windows mit der Registrierungsdatenbank einen zentralen Ort vor, indem die Daten binär und zum Teil komprimiert gespeichert sind [Wik21b] [Wik22g][YIRS17, S. 36ff][AIRS21, S. 391ff].

Zur Persistierung ist der Inhalt der Windows-Registry in binärer Form in mehreren Dateien, die als Hive-Files bezeichnet werden, verteilt gespeichert [AIRS21, S. 391ff].

Speicherorte der Dateien, in denen die Registrierungsdatenbank persistent gespeichert ist:

Schlüssel	Speicherort (Hive)
HKEY_LOCAL_MACHINE\BCD00000000 Boot-Konfiguration (BCD: Boot Configuration Database)	\Device\HarddiskVolume1\Boot\BCD
HKEY_LOCAL_MACHINE\COMPONENTS Konfiguration der CBS-Architektur (CBS: Component Based Servicing)	%systemroot%\System32\config\COMPONENTS
HKEY_LOCAL_MACHINE\HARDWARE Hardware-Konfiguration (wird bei jedem Boot neu aufgebaut)	-
HKEY_LOCAL_MACHINE\SAM Benutzerinformationen wie Anmeldenamen und Kennwörter (SAM: Security Accounts Manager)	%systemroot%\System32\config\SAM
HKEY_LOCAL_MACHINE\SECURITY systemweite Sicherheitsrichtlinien und Benutzerberechtigungen	%systemroot%\System32\config\SECURITY
HKEY_LOCAL_MACHINE\SOFTWARE Windows-Einstellungen, die nicht zum Booten benötigt werden, sowie Einstellungen installierter Anwendungsprogramme	%systemroot%\System32\config\SOFTWARE
HKEY_LOCAL_MACHINE\SYSTEM Windows-Einstellungen, die zum Booten benötigt werden	%systemroot%\System32\config\SYSTEM
HKEY_USERS\<user-sid>\ Einstellungen des Systems und der installierten Software, die sich auf ein spezielles Benutzerkonto beziehen	%systemdrive%\Users\<user-name>\NTUSER.DAT
(HKEY_CURRENT_USER\) Spiegelung von HKEY_USERS\<current-user-sid>\	-
(HKEY_CURRENT_CONFIG\) Spiegelung von HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Hardware Profiles\Current\	-
(HKEY_CLASSES_ROOT\) Spiegelung von HKEY_LOCAL_MACHINE\SOFTWARE\Classes\ und HKEY_CURRENT_USER\SOFTWARE\Classes\	-

Tabelle 1.7: Registry-Hives [Wik21b]

Die Datenbank ist hierarchisch organisiert und beinhaltet Container, die Registrierungsschlüssel (engl. Registry-Key) genannt werden. Jeder Schlüssel besitzt einen eindeutigen Bezeichner bzw. Namen und kann beliebig viele Kind-Schlüssel und Datenfelder enthalten. Zudem besitzt jeder Schlüssel einen Security-Deskriptor mit Besitz-Informationen und Access-Control-Liste (ACL), über die jedem Benutzer des Systems der Zugriff (lesend, schreibend) auf die Elemente eines Schlüssels und deren Kindelemente gewährt oder entzogen werden können. Die Datenfelder beinhalten die eigentlichen Konfigurationsdaten. Jedes Datenfeld besitzt wiederum einen eindeutigen Namen. Jeder Schlüssel

und jedes Datenfeld ist somit eindeutig über einen Pfad ähnlich einer Datei in einem Verzeichnissystem adressierbar [AIRS21, S. 391ff][YIRS17, S. 36ff][Wik22g]. Datenfelder weisen einen der folgenden sechs fest definierten Typen auf:

REG_DWORD	**32-Bit-Wert** 32-Bit Integer Wert
REG_QWORD	**64-Bit-Wert** 64-Bit Integer Wert
REG_BINARY	**Binärdaten** binärer Daten-Blob beliebiger Größe
REG_SZ	**Zeichenfolge** Unicode-Zeichenfolge (String) beliebiger Länge
REG_EXPAND_SZ	**Erweiterbare Zeichenfolge** Unicode-Zeichenfolge (String) beliebiger Länge, für die enthaltene Umgebungsvariablen (%SYSTEMROOT%) vom Configuration-Manager für einen lesenden Prozess automatisch expandiert werden
REG_MULTI_SZ	**Mehrteilige Zeichenfolge** beliebig lange Liste aus Unicode-Zeichenfolgen (Strings) beliebiger Länge, die als ein Multi-Parameter-String mit Standard-Trennzeichen abgelegt werden

Tabelle 1.8: Typen von Registry-Werten [Wik21b]

Der Configuration-Manager ist die zentrale Kernel-Komponente, welche die Zugriffe auf die Registry verwaltet. Über die Windows-API - im speziellen die `ADVAPI32.DLL` - werden Anwendungen Funktionen zur Verfügung gestellt, um Registry-Einträge direkt zu lesen, zu erstellen und zu bearbeiten. Neben dem direkten Zugriff auf einzelne Einträge führen viele andere Vorgänge der Windows-API indirekt zu Zugriffen auf die Registry. Die Funktionen führen dabei zu Aufrufen der `NTDLL.DLL` und schlussendlich zu Systemcalls, welche zum Configuration-Manager des Kernels geroutet werden. Dieser stellt die Informationen angefragter Schlüssel und Felder der Registrierungsdatenbank zur Verfügung oder modifiziert die Datenbank. Der Configuration-Manager greift durch Schnittstellen der kernelinternen File-System-Driver auf die Hive-Dateien des Massenspeichers zu, in denen die Registrierungsdatenbank persistent gespeichert wird. Da die internen Strukturen der Registrierungsdatenbank vom Configuration-Manager zwischengespeichert werden (Registry-Cache), führen bei weitem nicht alle Zugriffe auf die Registry zu Zugriffen auf die Hive-Files. Es ist nicht regulär vorgesehen, dass Anwendungen direkt auf die Hive-Dateien zugreifen, um Registry-Einträge zu lesen oder zu bearbeiten [Wik21b][Wik22g][AIRS21, S. 391ff].

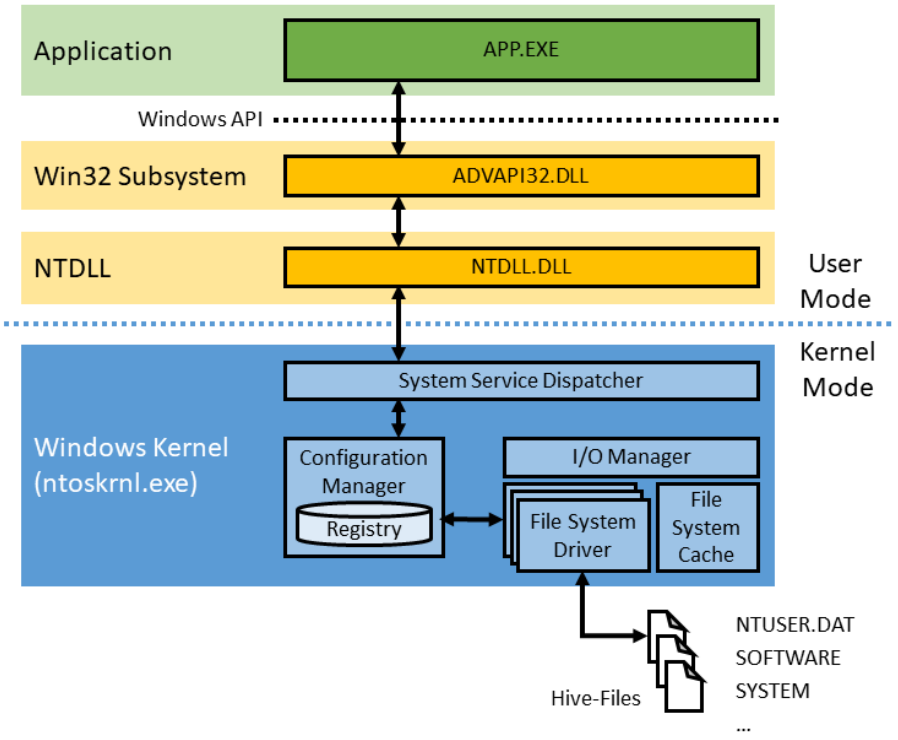

Abbildung 1.9: Zugriff auf die Windows-Registry

Das Lesen und Schreiben von Konfigurationsinformationen in der Windows-Registry stellt einen elementaren Verhaltensaspekt einer Software dar. Insbesondere Malware ändert böswillig die Systemkonfiguration zu eigenem Nutzen. Durch diese Änderungen versucht Malware das Verhalten des Systems und anderer installierter Software zu verändern. Solche Änderungen können verschieden Zwecken dienen. Beispiele wären das Stören von Sicherheitssoftware oder die eigene Persistierung. Über spezielle Schlüssel kann die Malware z.B. konfigurieren, dass sie bei der System-Initialisierung automatisch gestartet wird oder standardmäßig in jeden Adressraum geladen wird. Daher ist die Überwachung von Registry-Zugriffen auch im Kontext der Angriffserkennung als sehr relevant einzustufen.

Registry-Zugriffs-Funktionen der Windows-API

aus `AdvAPI32.DLL` (Advanced Win32 Base API)			
`RegCloseKey`	`RegOpenKey`	`RegConnectRegistry`	`RegOpenKeyEx`
`RegCreateKey`	`RegQueryInfoKey`	`RegCreateKeyEx`	`RegQueryMultipleValues`
`RegDeleteKey`	`RegQueryValue`	`RegDeleteValue`	`RegQueryValueEx`
`RegEnumKey`	`RegReplaceKey`	`RegEnumKeyEx`	`RegRestoreKey`
`RegEnumValue`	`RegSaveKey`	`RegFlushKey`	`RegSetKeySecurity`
`RegGetKeySecurity`	`RegSetValue`	`RegLoadKey`	`RegSetValueEx`
`RegNotifyChangeKeyValue`	`RegUnLoadKey`		

Tabelle 1.9: Registry-Zugriffsfunktionen der Windows-API [Wik21b]

reg key create	Registry-Schlüssel erstellt	ProcessID/GUID, ThreadID/GUID, Time, RegistryPath
reg key delete	Registry-Schlüssel gelöscht	ProcessID/GUID, ThreadID/GUID, Time, RegistryPath
reg key open	Registry-Schlüssel geöffnet	ProcessID/GUID, ThreadID/GUID, Time, KeyHandle, RegistryPath
reg key close	Registry-Schlüssel geschlossen	ProcessID/GUID, ThreadID/GUID, Time, KeyHandle
reg value create	Werte-Feld innerhalb des Schlüssels erstellt	ProcessID/GUID, ThreadID/GUID, Time, KeyHandle, valueName, valueType, initValue
reg value delete	Werte-Feld innerhalb des Schlüssels gelöscht	ProcessID/GUID, ThreadID/GUID, Time, KeyHandle, valueName
reg value read	Werte-Feld innerhalb des Schlüssels abgefragt	ProcessID/GUID, ThreadID/GUID, Time, KeyHandle, valueName, readValue
reg value write	Werte-Feld innerhalb des Schlüssels bearbeitet	ProcessID/GUID, ThreadID/GUID, Time, KeyHandle, valueName, setValue
reg key attribute read	Attribut des Schlüssel gelesen	ProcessID/GUID, ThreadID/GUID, Time, KeyHandle, attributeName, readValue
reg key attribute write	Attribut des Schlüssel geändert	ProcessID/GUID, ThreadID/GUID, Time, KeyHandle, attributeName, setValue

Tabelle 1.10: Generische Registry-Ereignistypen

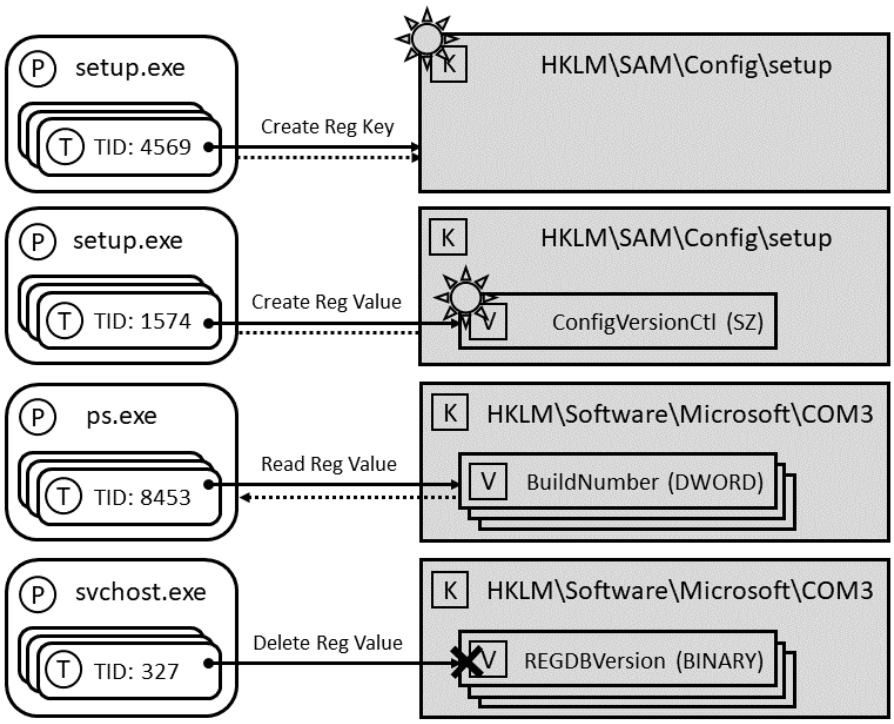

Abbildung 1.10: Generische Registry-Ereignistypen

1.5.6 Interprozesskommunikation (IPC)

Interprozesskommunikation (IPC) ist eine zentrale Funktionalität, die der Betriebssystem-Kernel den gehosteten Prozessen bereitstellt. Sie ermöglicht die Kommunikation eines Prozesses mit einem anderen Prozess. So existieren auf Computersystemen auch Service-Programme, die anderen Programmen Dienste bereitstellen. Diese werden nach dem Booten oder der Anmeldung des Benutzers gestartet und als Serviceprozesse dauerhaft ausgeführt. Ein Teil dieser Prozesse wartet auf Anfragen von Anwendungsprozessen oder anderen Serviceprozessen, um diese zu bearbeiten. Die Serviceprozesse übernehmen dabei die Rolle eines Servers, der anderen Prozessen, die als Client fungieren, Dienste bereitstellt. Die Kommunikation zwischen dem Serviceprozess und einem anfragenden Prozess wird über Kommunikationsschnittstellen der Interprozesskommunikation (IPC) umgesetzt. Auch Anwendungsprozesse können untereinander über IPC-Schnittstellen kommunizieren, die Rolle des Serviceprozess ist dafür nicht nötig. Im Zusammenhang mit Anfragen an Serviceprozesse ist Interprozesskommunikation jedoch vermehrt zu finden.

Der Begriff Interprozesskommunikation umfasst dabei sowohl Methoden zum Datentransfer zwischen Prozessen wie auch Methoden zur Synchronisation von Threads des selben oder verschiedener Prozesse [Wik20a].

Auch der Windows-Kernel bietet Prozessen verschiedene Möglichkeiten Interprozesskommunikation zu nutzen. Dazu zählen im Kontext Datentransfer die Verfahren Unnamed-Pipes, Named-Pipes, Message-Queues, (Advaced)-Local-Procedure-Calls (LPC/ALPC), Remote-Procedure-Calls (RPCs umgesetzt durch LPCs/ALPCs), Shared-Memory und einige weitere. Für die Synchronisation von Threads des gleichen oder verschiedener Prozesse werden die Methoden Mutex und Semphore angeboten. Auch unter Windows wird Interprozesskommunikation insbesondere Named-Pipes, ALPC, RPC und Message-Queues für die Kommunikation von Anwendungs-Prozessen mit Prozessen der Windows-Dienste (NT-Services) und installierten Diensten von Drittanbietern genutzt [AIRS21, S. 209ff][MicCe][Wik22b].

Ein interessanter Aspekt im Verhalten von Prozessen ist daher die Kommunikation (Synchronisation und Datentransfer) mit anderen Prozessen. Speziell Serviceanfragen von Prozessen an bestimmte Windows-Services können für die Nachvollziehbarkeit des Verhaltens sehr interessant sein. Daher ist eine Protokollierung von IPC-Ereignissen für die Überwachung des Prozessverhaltens relevant.

Ähnlich wie beim Zugriff auf Dateien, Verzeichnisse und Registry-Schlüssel werden die Ressourcen zur Interprozesskommunikation in Form von System-Objekten dem Prozess durch den Betriebssystem-Kernel bereitgestellt. Solche Objekte wie Named-Pipes, Message-Queues, RPC-Service-Provider, Mutexe und Semaphoren repräsentieren IPC-Datenstrukturen innerhalb des Kernels [AIRS21, S. 125ff][MicCe]. Dies sind z.B. Puffer

für Daten oder Nachrichten, Kommunikationskanäle, Verwaltungsstrukturen und vor Race-Conditions geschützte Zählvariablen für die Synchronisation. IPC-Objekte können von Prozessen unter einem Namen und Pfad im Object-Pool des, sich im Kernel befindlichen, Objekt-Managers angelegt werden und je nach Berechtigung von verschiedenen unterschiedlichen Prozessen gemeinsam verwendet werden [AIRS21, S. 125ff]. Dabei müssen diese Objekte in der Regel nach dem bereits beschriebenen Vorgehen zuerst geöffnet werden, wodurch der Kernel die Ressource über ein Handle bereitstellt, um danach mithilfe des Handels auf der bereitgestellten Ressource mit verschiedenen Funktionen zu operieren. Beispielsweise kann eine erstellte Named-Pipe von zwei Prozessen gleichzeitig geöffnet sein. Der eine Prozess schreibt Daten in die Named-Pipe, der andere kann diese aus der Named-Pipe lesen. Die Daten werden im Betriebssystem-Kernel in einem der Named-Pipe zugeordneten Pipe-Puffer zwischengespeichert.

IPC-Mechanismen zur Kommunikation
unamed pipes
named pipes
message queues
(advanced) local procedure calls (LPC/ALPC)
remote procedure calls (RPC) (nutzen ALPC)
shared memory

IPC-Mechanismen zur Synchronisation
mutex
semaphore

Tabelle 1.11: IPC-Mechanismen und IPC-Objekt-Typen

Manche Schadsoftware legt im Kontext der Infektionskette Mutexe an, um eigene Starts der Schadsoftware zu regulieren (verhindern, dass der Malware-Prozess mehrmals ausgeführt wird). Die angelegten Mutexe und deren Bezeichner können signifikant sein, um eine Infektion mit einer bestimmten Schadsoftware zu erkennen.

IPC object creat	neues IPC-Objekt anlegt	ProcessID/GUID, ThreadID/GUID, Time, IPCHandle, IPCType, ObjectPath, additionalParameter
IPC object delet	IPC-Objekt gelöscht	ProcessID/GUID, ThreadID/GUID, Time, ObjectPath, IPCType
IPC object open	IPC-Objekt geöffnet	ProcessID/GUID, ThreadID/GUID, Time, IPCHandle, IPCType, ObjectPath, additionalInfo
IPC object close	IPC-Objekt geschlossen	ProcessID/GUID, ThreadID/GUID, Time, IPCHandle, IPCType

Tabelle 1.12: Generische IPC-Ereignistypen (Teil 1)

pipe write	Daten in eine Pipe geschrieben	ProcessID/GUID, ThreadID/GUID, Time, PipeHandle, size, data, checksum
pipe read	Daten aus einer Pipe gelesen	ProcessID/GUID, ThreadID/GUID, Time, PipeHandle, size, data, checksum
message send	Nachricht gesendet	ProcessID/GUID, ThreadID/GUID, Time, MessageQueueHandle, size, data, checksum
message receive	Nachricht empfangen	ProcessID/GUID, ThreadID/GUID, Time, MessageQueueHandle, size, data, checksum
mutex lock	Mutex gesperrt	ProcessID/GUID, ThreadID/GUID, Time, MutexHandle
mutex unlock	Mutex freigegeben	ProcessID/GUID, ThreadID/GUID, Time, MutexHandle
semaphore acquire	P-Operation auf Semaphore anwendet (Wert verringert)	ProcessID/GUID, ThreadID/GUID, Time, SemaphoreHandle
semaphore release	V-Operation auf Semaphore anwendet (Wert erhöht)	ProcessID/GUID, ThreadID/GUID, Time, SemaphoreHandle

Tabelle 1.13: Generische IPC-Ereignistypen (Teil 2)

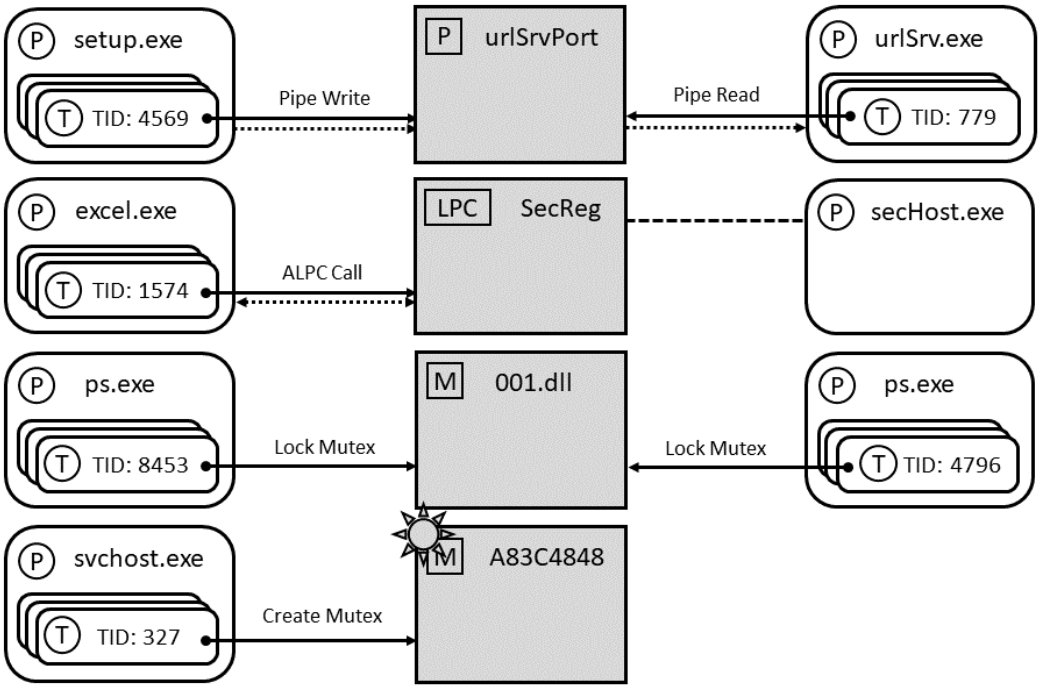

Abbildung 1.11: Generische IPC-Ereignistypen

1.5.7 Interaktion mit dem Endanwender

Die Interaktion mit dem Endanwender ist allgemein betrachtet ein bedeutender Teil des Verhaltens eines Prozesses. So nimmt dieser für interaktive Anwendungen mit grafischer Benutzeroberfläche meist den größten Anteil am Prozessverhalten ein. Da dieser jedoch für die Malware- und Angriffserkennung weniger bedeutend ist, soll hier nur eine kurze Erläuterung gegeben werden und keine Ereignisse spezifiziert werden.

Nicht alle Prozesse interagieren mit dem Endanwender. Gerade im Serverbetrieb erfüllen die Prozesse selbstständig ihre Aufgaben ohne dass eine Interaktion mit dem Benutzer notwendig ist. Auch auf einem interaktiven PC-System, an dem ein Anwender arbeitet, existieren viele System- und Hintergrundprozesse ohne Nutzerinteraktion. Die Prozesse, die Nutzerinteraktion durchführen, lassen sich in zwei Gruppen einteilen, Konsolen-Anwendungen und Anwendungen mit grafischer Benutzeroberfläche.

Die erst genannten kommunizieren über textuelle Bildschirm-Ein- und Ausgaben mit dem Endanwender. Diese werden über das Schreiben in ein bzw. zwei Ausgabe-Datenströme des Prozesses realisiert (`stdout`: normale Ausgabe, `stderr`: Fehlerausgabe). Diese sind entweder als zwei private Unnamed-Pipe-Objekte oder als ein Konsolen-Objekt realisiert. Letzteres stellt drei textuelle Kommunikationskanäle zu einem Terminal-Prozess zur Verfügung (`ConDrv`, `ConHost`). Auf der Gegenseite der Pipe oder des Konsolen-Objektes liest der Elternprozess oder ein Terminal-Prozess die Ausgaben und bringt diese auf den Bildschirm. Die Eingaben werden in gleicher Weise umgesetzt, indem der Prozess aus dem Datenstrom `stdin` liest, indem die Gegenseite die Tastatureingaben des Benutzers hineingeschrieben hat. Soll diese Interaktion bezüglich einer Ereignisprotokollierung überwacht werden, müssten Event-Objekte für die Ereignisse des Lesens aus `stdin` und des Schreibens in `stdout` / `stderr` erzeugt und protokolliert werden [MicCg].

Für die zweit genannte Gruppe, den Anwendungen mit grafischer Benutzeroberfläche, stellen die Interaktionsereignisse bezüglich der Benutzerinteraktion in den meisten Fällen den weit größten Teil des gesamten Prozessverhaltens dar. Prozesse die eine GUI anzeigen möchten, registrieren dafür ein oder mehrere Fenster beim Fenstermanager. Anders als bei UNIX/Linux war bei Windows der Fenstermanager im Betriebssystem-Kernel lokalisiert. Seit Windows Vista ist dieser in den Prozess DWM (desktop window manager) ausgelagert, sodass der Großteil der Funktionalität im User-Mode ausgeführt wird. Das Kernel-Interface, das Prozesse zur Registrierung und Steuerung ihrer Fenster benutzen, besteht aber nach wie vor im Windows-Kernel und leitet die Anfragen an den DWM-Prozess weiter. Somit haben die einzelnen Prozesse keinen direkte Interprozesskommunikation mit dem DWM-Prozess [Wik22a]. Die Prozesse können über das Kernel-Interface GDI (graphic display interface) auf den von ihnen registrierten Fenstern zeichnen und somit die Benutzeroberfläche mit Steuerelementen und anderem quasi

auf die Fensteroberfläche zeichnen. Zudem existiert für jedes Fenster mindestens eine Message-Queue, über die das Fenstermanager-Interface im Kernel Benutzeraktionen dem Prozess als Event-Nachrichten sendet. Bewegt sich der Cursor über ein Fenster oder klickt in ein Fenster wird beispielsweise dem Prozess, der das Fenster registriert hat, eine Nachricht geschickt, die Aktion zu behandeln. Tastaturanschläge werden dem Prozess gemeldet, dessen registriertes Fenster aktuell im Fokus ist. Eine Ereignisprotokollierung müsste Aktionen an dieser Schnittstelle zwischen Prozess und Fenstermanager aufzeichnen. Da diese Aktionen sehr kleinteilig sind, würde sich dies sehr schwierig gestalten. Es wären vermutlich gute Vorfilterungen und Aggregation notwendig, um brauchbare Informationen zu generieren und nicht durch eine extrem großen Menge an GUI-Ereignissen überlastet zu sein. Ein großer Teil der Gesamtmenge an Systemcalls auf einem Windows-System kann auf die Kommunikation zwischen Prozessen mit grafischer Benutzeroberfläche und dem Fenstermanager-Kernel-Interface zurückgeführt werden (sieht Kapitel Auftrittsfrequenzen).

1.5.8 Netzwerk-Kommunikation

Ein wichtiger Aspekt bei der Überwachung des Prozessverhaltens ist der Zugriff auf das Netzwerk und Ressourcen im Internet. Der dabei involvierte Netzwerk-Protokoll-Stapel (TCP/IP-Stack) ist bei Windows wie auch bei Linux zum größten Teil im Betriebssystem-Kernel implementiert. Unter Windows betrifft das konkret den größten Teil der Transportschicht (TCP/UDP-Backend) und die gesamte Vermittlungsschicht (IP). Die darunter liegenden Zugriffsschichten (Media Access Layer) werden von im Kernel befindlichen Filtertreibern (NDIS-Treiber) und den Gerätetreibern (MAC- und NIC-Treiber) der Netzwerkadapter umgesetzt. Der Betriebssystem-Kernel stellt dem Prozess Netzwerk-Kommunikationsmöglichkeiten bereit, die dieser nutzen kann, um mit anderen Prozessen auf entfernten Rechnern über das Internet zu kommunizieren. In der Regel geschieht dies in Form von Anwendungsprotokollen (DNS, HTTP(S), FTP(S), SMTP, RTMP, XMPP, DHCP und viele weitere). Diese kann eine Anwendung selbst implementieren. Fast immer werden dafür jedoch Bibliotheken der Windows-API oder von Dritt-Anbietern eingesetzt. Werden Funktionalitäten der Windows-API genutzt, können auch Windows-Services in die Kommunikation involviert sein (z.B. DNSCACHE-Service).

Der Zugriff auf die Netzwerk-Funktionalität (TCP/UDP-Frontend) des Kernels wird dem Prozess über Socket-Objekte (Windows-Sockets kurz WinSock) durch den Betriebssystem-Kernel bereitgestellt.

Internetprotokoll (IP)

IP packet send	IP-Paket versendet	ProcessID/GUID, ThreadID/GUID, Time, packetID/GUID, parentProtocolActionID/GUID, sourceAddr, destAddr, size, data, checksum
IP packet receive	IP-Paket empfangen	ProcessID/GUID, ThreadID/GUID, Time, packetID/GUID, parentProtocolActionID/GUID, sourceAddr, destAddr, size, data, checksum
ICMP request	ICMP-Request gesendet	ProcessID/GUID, ThreadID/GUID, Time, packetID/GUID, ICMP-Typ, ICMP-Code, ICMP-Daten
ICMP response	ICMP-Request empfangen	ProcessID/GUID, ThreadID/GUID, Time, packetID/GUID, ICMP-Typ, ICMP-Code, ICMP-Daten

TCP-Transportprotokoll

TCP connection request	TCP-Verbindungsaufbau-Anfrage gesendet	ProcessID/GUID, ThreadID/GUID, Time, segmentID/GUID, parentProtocolActionID/GUID, sourceIpAddr, sourcePort, destIpAddr, destPort
TCP disconnection request	TCP-Verbindungsabbau-Anfrage gesendet	ProcessID/GUID, ThreadID/GUID, Time, segmentID/GUID, parentProtocolActionID/GUID, connectionID/GUID
TCP listen	warten auf eine TCP-Verbindung	ProcessID/GUID, ThreadID/GUID, Time, tcpListenID/GUID, parentProtocolActionID/GUID ipAddr, port
TCP listen end	Warten auf eine TCP-Verbindung beendet	ProcessID/GUID, ThreadID/GUID, Time, tcpListenID/GUID, parentProtocolActionID/GUID
TCP connected	TCP-Verbindung aufgebaut	ProcessID/GUID, ThreadID/GUID, Time, parentProtocolActionID/GUID, connectionID/GUID
TCP disconnected	TCP-Verbindung beendet	ProcessID/GUID, ThreadID/GUID, Time, parentProtocolActionID/GUID, connectionID/GUID
TCP send	TCP-Segment gesendet	ProcessID/GUID, ThreadID/GUID, Time, packetID/GUID, connectionID/GUID, parentProtocolActionID/GUID, segmentNumber, size, data, checksum
TCP receive	TCP-Segment empfangen	ProcessID/GUID, ThreadID/GUID, Time, packetID/GUID, connectionID/GUID, parentProtocolActionID/GUID, segmentNumber, size, data, checksum

Tabelle 1.14: Generische Netzwerk-Ereignistypen (Teil 1)

UDP-Transportprotokoll

UDP send	UDP-Datagram gesendet	ProcessID/GUID, ThreadID/GUID, Time, packetID/GUID, parentProtocolActionID/GUID, sourceIPAddr, sourcePort, destIpAddr, destPort, size, data, checksum
UDP receive	UDP-Datagram empfangen	ProcessID/GUID, ThreadID/GUID, Time, packetID/GUID, parentProtocolActionID/GUID, sourceIPAddr, sourcePort, destIpAddr, destPort, size, data, checksum

Sitzungsprotokolle

... TLS Verschlüsselung usw.

Anwendungsprotokolle

HTTP request	HTTP-Anfrage gesendet	ProcessID/GUID, ThreadID/GUID, Time, hostname, requestMessage, clientInfo, protocolVersion
HTTP response	HTTP-Antwort empfangen	ProcessID/GUID, ThreadID/GUID, Time, hostname, responseCode, serverInfo, protocolVersion, contentLength, contentType, content, checksum
DNS request	DNS-Anfrage gesendet	ProcessID/GUID, ThreadID/GUID, Time, hostname, type, fullContent
DNS response	DNS-Antwort emfpangen	ProcessID/GUID, ThreadID/GUID, Time, ipAddr, authorityInfo fullContent

... Ereignisse noch vieler weiterer Anwendungsprotokolle

Tabelle 1.15: Generische Netzwerk-Ereignistypen (Teil 2)

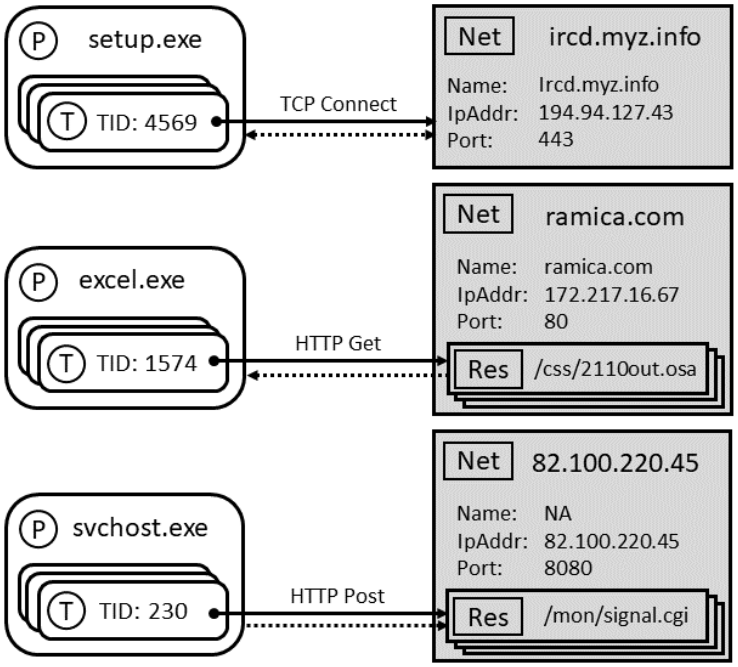

Abbildung 1.12: Generische Netzwerk-Ereignistypen

1.5.9 Gerätenutzung

Ein weiterer Aspekt in der Überwachung des Prozessverhaltens stellt der Zugriff auf IO-Geräte und virtuelle Software-Geräte dar. Dahinter steckt die Kommunikation des Prozesses mit dem im Kernel befindlichen Gerätetreiber der Hardware. Treiber können für ihre Geräte, die sie steuern, Geräte-Objekte im Objekt-Pool des Objekt-Managers anlegen. Prozesse können über Funktionalitäten des Betriebssystem-Kernels auf diese Geräte-Objekte zugreifen und dadurch das IO-Gerät oder ein virtuelles Software-Gerät verwenden. Die Anfragen, die der Prozess an das Gerät stellt, werden an den dahinter liegenden Treiber, der das Geräte-Objekt registriert hat, weitergeleitet, damit dieser diese bearbeitet.

Zwar sind IO-Geräte indirekt sehr häufig in das Verhalten eines Prozesses verwickelt, jedoch nur selten direkt. Beispielsweise führen Datei-Zugriffe, Registry-Zugriffe oder Netzwerk-Zugriffe unter Umständen zu IO-Vorgängen auf den Datenträger-Geräten (Storage-Devices) oder den Netzwerkadaptern (NICs). Jedoch geschehen diese Zugriffe nicht direkt, sondern indirekt über Zwischeninstanzen wie dem Datei-Manager, dem Config-Manager oder dem Network-Stack. Das ein direkter Zugriff auf IO-Geräte von Malware durchgeführt wird, ist sehr selten. Jedoch gibt es vereinzelte Szenarien im Kontext der Angriffserkennung, die eine Überwachung der Gerätenutzung relevant erscheinen lassen. Dies könnte z.B. in komplexen Angriffen, die auf einem hohen technischen Niveau ablaufen, der Fall sein. Hier könnten z.B. physische IO-Geräte Ziel des Angriffes sein. Beispielsweise könnte die Malware versuchen die Firmware der Geräte zu infizieren, um sich zu persistieren oder zu verbreiten. Ein anderes Angriffsziel könnten die Gerätetreiber bestimmter Geräte sein. Eine Exploitation eines Gerätetreibers über die Geräteschnittstelle (Geräte-Objekt) durch einen bösartigen Prozess, könnte diesen in die Lage versetzen, Code durch den Treiber im Kernel-Mode ausführen zu lassen oder anderweitig Speicher- und Ressourcenschutz-Mechanismen zu umgehen.

Für die Überwachung der Geräte- bzw. Treiberschnittstellen sind folgende allgemeine Ereignisse zu nennen:

device open	Geräte-Objekt geöffnet	ProcessID/GUID, ThreadID/GUID, Time, DeviceHandle, DeviceObjectPath
device close	Geräte-Objekt geschlossen	ProcessID/GUID, ThreadID/GUID, Time, DeviceHandle
device function request (IOCTL)	Gerätefunktion aufgerufen	ProcessID/GUID, ThreadID/GUID, Time, DeviceHandle, MajorDeviceFunction, MinorDeviceFunction, dataSize, data, checksum

Tabelle 1.16: Generische IO-Ereignistypen

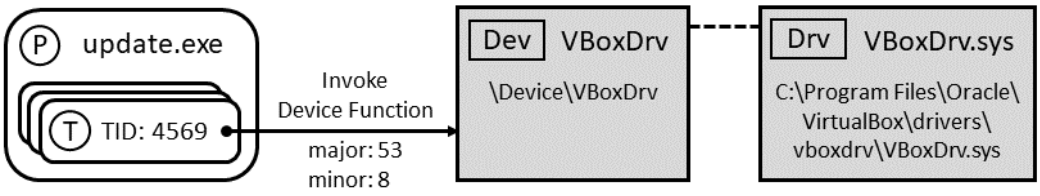

Abbildung 1.13: Generische IO-Ereignistypen

1.5.10 Zugriffe auf fremde Prozesse

Es existieren legitime Anwendungsfälle, bei denen ein Prozess Zugriff auf die geschützten Ressourcen eines anderen Prozesses benötigt. Beispielsweise muss der Prozess einer Debugger- oder Analysesoftware auf den Adressraum und die Verwaltungsstrukturen des zu debuggenden- bzw. zu analysierenden Prozesses zugreifen können, um in dessen Code Haltepunkt einzufügen, die Speicher- und Registerinhalte auszulesen und zu manipulieren. Neben diesem plakativen Beispiel können viele weitere Anwendungsfälle gefunden werden auch abseits der Softwareentwicklung. Betriebssysteme sind daher gefordert, Möglichkeiten zu schaffen, wodurch ein Prozess auf die Inhalte des geschützten virtuellen Speicher und die Verwaltungsstrukturen eines anderen Prozesses zugreifen und manipulieren kann. Auch Windows bietet verschiedene Betriebssystemfunktionen die dies ermöglichen. Sehr wichtig ist jedoch, dass ein solcher Zugriff durch Zugriffsberechtigungen stark reguliert wird, sodass nicht jeder beliebige Prozess auf jeden beliebigen anderen Prozess zugreifen kann. Wäre dies möglich, wäre jeglicher Speicher- und Ressourcenschutz sinnlos. Stattdessen regeln Zugriffsberechtigungen, welcher Prozess auf welche anderen Prozesse zugreifen darf. Dies geschieht über Security-Deskriptoren mit Access-Control-Listen (ACLs) der Prozess- und Thread-Objekte. Aus verschiedenen Gründen, z.B. der Abwärtskompatibilität, gestaltet sich die Vergabe von Zugriffsberechtigungen unter Windows meist recht grob, was dazu führt, dass standardmäßig viele Systemprozesse die Berechtigung besitzen, solche Aktionen durchzuführen, die diese nicht benötigen sollten. So darf z.B. standardmäßig jeder mit Administratorberechtigungen gestartete Prozess auf fremde Prozesse zugreifen. Niedriger privilegierte Prozesse, z.B. durch den Benutzer gestartete Anwendungsprogramme, dürfen auf die meisten ihrer Kindprozesse zugreifen [Wik21a]. Zusammenfassend muss gesagt werden, dass die Regulierung dieser Funktionalitäten durch Zugriffsberechtigungen unter Windows in der Vergangenheit eine nicht zufriedenstellende Wirkung aufwies. Aus diesem Grund wurde diese Funktionalität stark von Schadsoftware genutzt. Beispiele sind das Stehlen von geheimen Daten wie beispielsweise Zugangsdaten und Passwörtern aus den Adressräumen fremder Prozesse, das manipulieren von fremden Prozessen zum eigenen Nutzen oder das Einschleusen von Schadsoftware in fremde Prozesse, um die Kontrolle über diese zu übernehmen und

schadhafte Vorgänge aus einem vermeintlich legitimen Prozess heraus durchzuführen [Hos17].

Ereignisse die Zugriffe auf fremde Prozesse anzeigen, stellen daher einen sehr wichtigen Teil bei der Verhaltensüberwachung von Prozessen dar. Diese Ereignisse sind zudem im besonderen Maße für die Erkennung von Schadsoftware und Angriffen relevant, da sie maßgeblich von Schadsoftware genutzt werden und zudem eine Schlüsselrolle für die Funktionalität bestimmter Malware und beim Verfolgen von Angriffen darstellen.

Für den Zugriff auf fremde Prozesse können folgende generische Ereignisse spezifiziert werden:

process open	Prozess-Objekt eines fremden Prozesses öffnen	ProcessID/GUID, ThreadID/GUID, Time, processHandle, foreignProcessID/GUID
process close	Prozess-Objekt eines fremden Prozesses schließen	ProcessID/GUID, ThreadID/GUID, Time, processHandle
thread open	Thread-Objekt eines fremden Prozesses öffnen	ProcessID/GUID, ThreadID/GUID, Time, threadHandle, foreignProcessID/GUID, foreignThreadID/GUID
thread close	Thread-Objekt eines fremden Prozesses schließen	ProcessID/GUID, ThreadID/GUID, Time, threadHandle
process memory read	Daten aus dem Adressraum eines fremden Prozesses lesen	ProcessID/GUID, ThreadID/GUID, Time, processHandle, address, data, size, checksum
process memory write	Daten in den Adressraum eines fremden Prozesses schreiben	ProcessID/GUID, ThreadID/GUID, Time, processHandle, address, data, size, checksum
process attribute read	Attribut eines fremden Prozesses abfragen	ProcessID/GUID, ThreadID/GUID, Time, processHandle, attribute, value
process attribute write	Attribut eines fremden Prozesses verändern	ProcessID/GUID, ThreadID/GUID, Time, processHandle, attribute, value
thread attribute read	Attribut eines fremden Threads abfragen (z.B. Registerinhalte)	ProcessID/GUID, ThreadID/GUID, Time, threadHandle, attribute, value
thread attribute write	Attribut eines fremden Threads ändern (z.B. Registerinhalte)	ProcessID/GUID, ThreadID/GUID, Time, threadHandle, attribute, value

Tabelle 1.17: Generische Fremd-Prozess-Zugriff-Ereignistypen (Teil 1)

process memory map	Etwas in den Adressraum eines fremden Prozesses einblenden	ProcessID/GUID, ThreadID/GUID, Time, processHandle, baseAddress, image
remote thread create	Remote-Thread im fremden Prozess erstellen	ProcessID/GUID, ThreadID/GUID, Time, processHandle, threadHandle, createdThreadID/GUID, startAddr, stackBaseAddr
remote thread start	Remote-Thread im fremden Prozess starten	ProcessID/GUID, ThreadID/GUID, Time, threadHandle
remote thread stop	Remote-Thread im fremden Prozess stoppen	ProcessID/GUID, ThreadID/GUID, Time, threadHandle

Tabelle 1.18: Generische Fremd-Prozess-Zugriff-Ereignistypen (Teil 2)

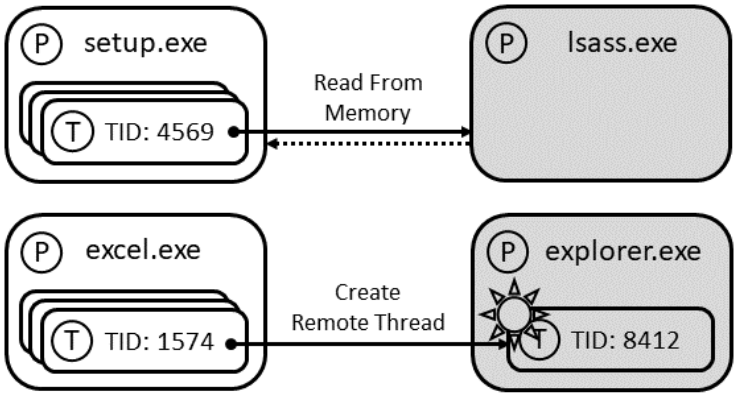

Abbildung 1.14: Generische Fremd-Prozess-Zugriff-Ereignistypen

1.5.11 Laden von Kernel-Modulen

Prozesse können den Betriebssystem-Kernel veranlassen, dass dieser ein Kernel-Modul bzw. Kernel-Treiber lädt. Unter Windows liegen kompilierte Kernel-Treiber als PE-Executables in SYS-Dateien (.sys) vor. Die Speicherbereiche werden beim Laden in den Adressraum des Kernel eingeblendet. Der Kernel-Code kann mithilfe der Referenzen exportierter Symbole in Funktionen des geladenen Treibers einspringen und die Funktionalität verwenden. Über diese Methode bedient der Treiber-Code die Treiber- und Geräteschnittstellen des Kernels [YIRS17, S. 564ff][Yos20, S. 64ff]. Für die verschiedenen Basisfunktionen eines Treibers, wie Initialisierung und das Entgegennehmen von Anfragen bezüglich der Gerätesteuerung sind eine feste Menge an vordefinierten, erwarteten Symbolen durch das Treiber-Image exportiert. Der Code eines geladenen Kernel-Moduls wird im Kernel-Mode ausgeführt. Der Thread- und Prozesskontext kann dabei verschieden sein, je nachdem

in welchem Kontext Funktionen des Treibers verwendet werden (Anfragen gestellt werden). Treiber besitzen die Möglichkeit Callback-Funktionen im Kernel zu registrieren, sodass ihr Code als Reaktion auf bestimmte Ereignisse im Kernel aufgerufen wird und direkt (synchron) ohne Verzögerung ausgeführt wird. Der Kontext wird in dem Fall durch den Thread bestimmt, der das Ereignis ausgelöst hat [Bau21]. Speziell können sie auch Funktionen als Interrupt-Service-Routinen (ISR) für bestimmte Interrupt-Vektoren beim Kernel registrieren, die der Interrupt-Dispatcher des Kernels aufruft, wenn ein passender Interrupt eingeht. Auch in diesem Fall ist der Thread und Prozesskontext nicht allgemein vorhersagbar. Da der Kernel-Code im Kernel-Mode ausgeführt wird, ist der Treiber-Code prinzipiell in der Lage, die Hardware-Plattform uneingeschränkt zu steuern. Speicherschutz-Mechanismen, wie sie für User-Mode-Code gelten, bestehen entweder nicht (für Adressraum des aktuellen Prozess-Kontextes) oder können umgangen werden (für alle Adressräume). Insofern hat der Code eines Kernel-Moduls prinzipiell einen Vollzugriff auf alle Adressräume des Systems und somit auf alle Speicherbereiche aller Prozesse.

Ein bösartiges Kernel-Modul wäre in der Lage, Daten aus den Speicherbereichen beliebiger Prozesse auszulesen. Über solche Methoden könnten beispielsweise eingegebene Passwörter oder andere schützenswerte Informationen gestohlen werden. Es ist zudem auch möglich, Daten in die Speicherbereiche zu schreiben. Dadurch können Prozesse wie auch das Betriebssystem manipuliert werden. Über diese Methode kann die Schadsoftware sich in weitere Komponenten wie andere Prozesse und Kernel-Bestandteile ausbreiten und somit das System auf einer sehr tiefen Ebene kompromittieren. Z.B. könnte das bösartige Kernel-Modul versuchen, die Berechtigungen eines speziellen bösartigen User-Mode-Prozesses auszuweiten. Aktuelle Windows-Version besitzen Gegenmaßnahmen, um diese Form von Angriffen abzuwehren. Zum einen müssen im regulären Betrieb Treiber-Executables von Microsoft digital signiert sein. Beim Ladevorgang prüft der Kernel die Signatur und verweigert das Laden bei nicht signierten Modulen [YIRS17, S. 661ff][Yos20, S. 245ff]. Firmen die Kernel-Module entwickeln, müssen diese durch Microsoft zertifizieren lassen, welches eine Prüfung beinhaltet, die sicherstellen soll, dass Kernel-Treiber keine illegitime oder schadhafte Wirkung haben [Yos20, S. 345ff]. Windows kann über die Boot-Konfiguration so konfiguriert werden, dass es in einem Entwicklermodus startet. In diesem können auch unsignierte Treiber geladen werden [YIRS17, S. 661ff]. Aufgrund einer fehlenden Signatur werden in Angriffen daher selten direkt Malware-Treiber eingesetzt. Stattdessen veranlasst eine Schadsoftware das Laden legitimer, signierter Gerätetreiber, die den Angreifern bekannte Schwachstellen aufweisen, die nach dem Laden über die Treiber- oder Geräte-Objekt-Schnittstelle von der Schadsoftware angegriffen werden. Die im User-Mode ausgeführte Malware versucht über die Kommunikationsschnittstellen zum Treiber diesen zu exploitieren und darüber den Kernel zu manipulieren, Informationen zu stehlen oder Code in den angegriffenen Treiber einzu-

schleusen und im Kernel-Mode zur Ausführung zu bringen. Ein weiterer Abwehrmechanismus im Windows-Kernel stellt die Kernel-Patch-Protection (KPP), auch PatchGuard genannt, dar. Dabei handelt es sich um eine in Intervallen durchgeführte Integritätsprüfung der meisten Daten des Kernels (gesamter Kernel-Code ausgenommen geladener Treiber und viele Datenstrukturen). Stellt PatchGuard eine Integritätsverletzung fest, indem die Daten eines Bereiches nicht mit einer Checksumme übereinstimmen, wird das System mit einem Bluescreen-of-Death (BSOF) beendet. Auch PatchGuard ist im Entwicklermodus nicht aktiv [YIRS17, S. 890ff]. Kernel-Patch-Protection (KPP) schränkt damit die Angriffsmöglichkeiten bösartiger Kernel-Module ein, kann diese jedoch nicht vollständig verhindern, sodass nach wie vor eine große Gefahr von bösartigen Code, der im Kernel-Mode ausgeführt wird, ausgeht.

Aus diesem Grund ist die Überwachung von Treiber-Lade-Aufforderung von Prozessen für die Angriffserkennung sehr relevant. Das Prozesse das Laden von Kernel-Modulen veranlassen ist ein seltener Vorgang. Er kann legitim bei der Installation von Gerätetreibern von der Installationssoftware eingesetzt werden, oder beim Einbringen von Monitoring-Modulen (z.B. Filter-Treiber) durch Sicherheits- oder Systemsoftware. Darüber hinaus lädt, startet und stoppt der Filter-Manager-Prozess Filter-Treiber.

Für das Laden von Kernel-Modulen können somit zwei generische Ereignisse spezifiziert werden:

kernel module load	Kernel-Modul geladen	ThreadID/GUID, ProcessID/GUID, Time, ImageName, FileName/Path, Checksum, ImageBaseAddr, SignatureInformation
kernel module unload	Kernel-Modul entladen	ThreadID/GUID, ProzessID/GUID, Time, ImageName

Tabelle 1.19: Generische Kernel-Modul-Load-Ereignistypen

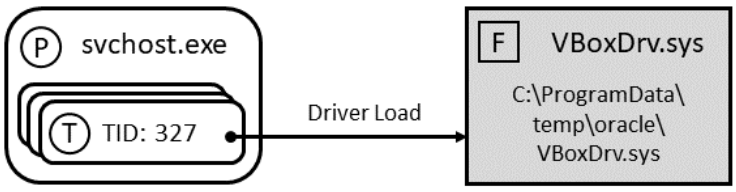

Abbildung 1.15: Generische Kernel-Modul-Load-Ereignistypen

1.6 Einfache Aggregationen

Es existieren bekannte Verhaltensmuster, für die sich eine Sequenz aus mehreren Ereignissen der Prozessverhaltensebene einfach zu einem aggregierten logischen Ereignis zusammenfassen lässt. In diesem Abschnitt sollen drei unterschiedliche Typen solcher einfacher Aggregation anhand von Beispielen erläutert werden.

1.6.1 Aggregation von Ereignissequenzen

Einige Vorgänge im Verhalten von Prozessen bestehen aus Sequenzen von Unterereignissen. Ein klassisches und plakatives Beispiel stellt der lesende und schreibende Zugriff auf Dateien dar. Dabei wird beim Lesen, wie bereits in vorangegangenen Abschnitten beschrieben, zuerst die Datei geöffnet. Dies resultiert in einem `OpenFile`-Ereignis. Dem folgen Leseereignisse, welche anzeigen, dass Daten aus der Datei in den Speicherbereich des Prozesses transferiert wurden. Schlussendlich schließt der Prozess die Datei wieder, was in einem `CloseFile`-Ereignis resultiert. Eine solche Ereignissequenz kann zu einem einzelnen `ReadFile`-Ereignis zusammengefasst werden, welches anzeigt, dass Daten aus der Datei gelesen wurden. Interessant ist die Frage nach dem Zeitpunkt dieses aggregierten Ereignisses. Liegt dieser auf dem Auftrittszeitpunkt von `CloseFile` wird dadurch der Abschluss der Sequenz markiert. Alternativ könnte auch der Zeitpunkt gewählt werden, an dem dem Prozess die ersten Daten im Speicher zur Verfügung stehen. Dann müsste der Zeitpunkt des ersten `Read`-Ereignisses gewählt werden. Eine Datei kann von einem Prozess über eine lange Zeit geöffnet sein und es kann zu verschiedenen Zeitpunkten, die auch zeitlich entfernt sein können, gelesen werden. Um dies zu berücksichtigen, könnten z.B. nur `Read`-Ereignisse, die zeitlich nah beieinander liegen, zusammengefasst werden, sodass auch nach der Aggregation eventuell noch mehrere Zugriffsereignisse auf dieselbe Datei vorhanden sind. Die Art der Aggregation hängt vom Anwendungsfall ab. Konkret stellt sich die Frage, ob es ausreicht zu wissen, dass ein Prozess allgemein Daten aus einer speziellen Datei gelesen hat oder ob es auch wichtig ist, wann, in welchen Intervallen und welche Daten aus einer speziellen Datei gelesen wurden.

Ähnlich wie beim Lesevorgang verhält es sich auch bei Schreibvorgängen auf eine Datei. Auch hier besteht die Sequenz aus Unterereignissen beginnend mit dem Öffnen. Darauf folgen Schreibereignisse, die anzeigen, dass ein Pufferinhalt vom Speicherbereich des Programms in den Speicherbereich des Betriebssystem-Kernels kopiert wurde, damit dieser ihn in die Datei schreiben kann, bzw. dem Dateisystem-Treiber übergeben kann. Abschließend schließt der Prozess die Datei, was in einem `CloseFile`-Ereignis resultieren kann. Beim Schreibvorgang existiert zudem der Flush-Vorgang, der durch `Flush`-Ereignisse signalisiert wird. Diese können explizit durch entsprechende Funktionsaufrufe durch den Prozess verursacht werden, oder implizit in den tiefer liegenden

Schichten, die für die Umsetzung des Schreibprozesses verantwortlich sind. In diesen Schichten, wie der Laufzeitbibliotheken, dem Betriebssystem-Kernel, dem Dateisystem-Treiber, werden zu schreibende Daten gepuffert und zu späteren Zeitpunkten tatsächlich in die Datei geschrieben. Das Flush-Ereignis signalisiert, dass tatsächlich Daten in die Zieldatei geschrieben wurden. Dies geschieht explizit auf Anweisung durch die Software, implizit zu anderen günstigen Zeitpunkten, bei einem vollen Schreibpuffer oder spätestens direkt bevor die Datei geschlossen wird. Der Zeitpunkt des ersten Flush-Ereignissen markiert somit den frühsten Zeitpunkt, an dem die Datei durch den Prozess in dieser Aktion eine Änderung erfahren hat und kann somit als Zeitpunkt für ein aggregiertes `WriteFile`-Ereignis herangezogen werden. Ein wichtiges Attribut einer jeden Datei ist der Modifikationszeitpunkt, also der Zeitpunkt der letzten stattgefunden Änderung des Dateiinhalts. Diese Eigenschaft ergibt sich durch den letzten Flush-Zeitpunkt. Dieser stellt somit auch einen interessanten Zeitpunkt für ein aggregiertes `WriteFile`-Ereignis dar [Sol20]. Auch beim Zusammenfassen mehrerer Unterereignisse der Aktionsfrequenz zum Schreiben in eine Datei wirft dieselben Fragen auf, wie sie auch für den Vorgang des Lesen aus der Datei beschrieben wurden. So stellt sich auch hier die Frage, ob ein Zusammenfassen der Vorgänge auch dann sinnvoll sind, wenn die Datei über einen langen Zeitraum geöffnet ist und in größeren zeitlichen Abständen geschrieben wird.

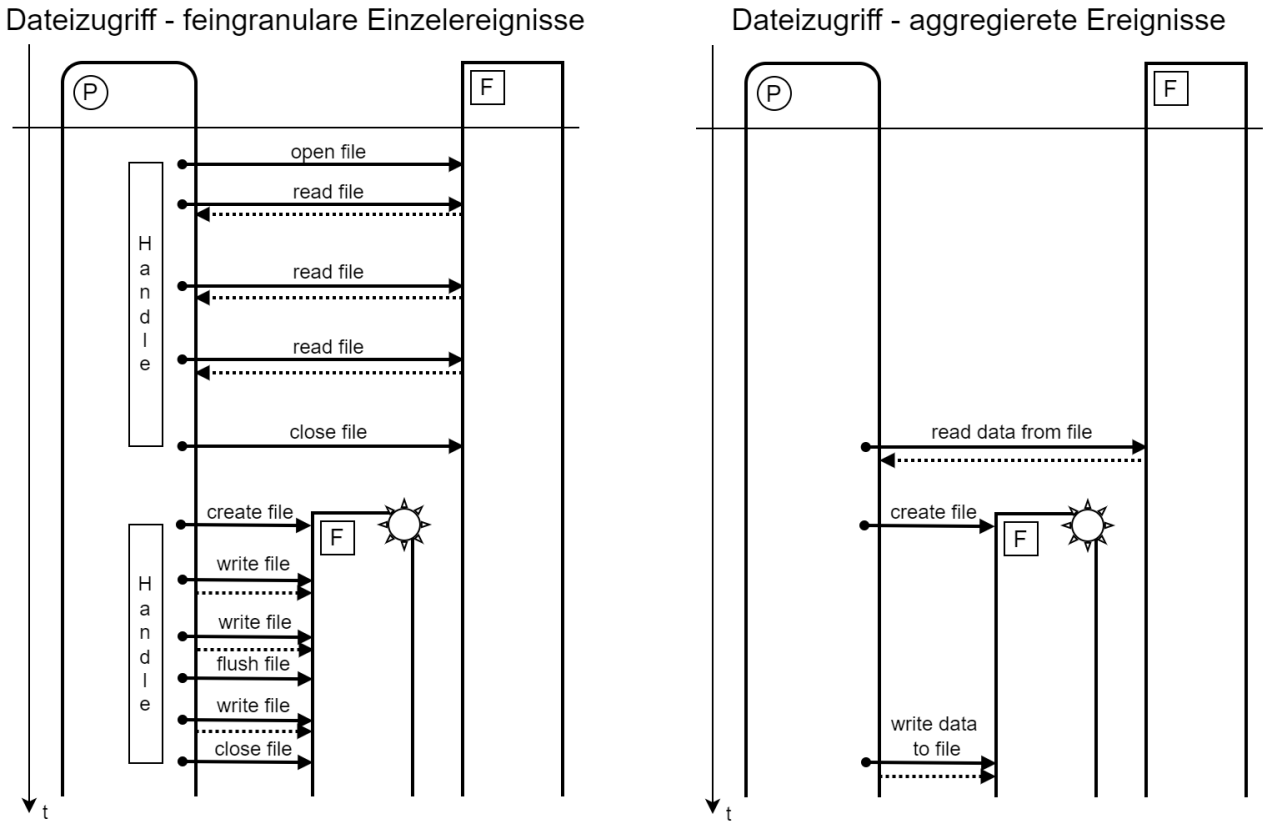

Abbildung 1.16: Aggregation von Ereignissequenzen am Beispiel Dateizugriffe

1.6.2 Aggregation von Ereignissen aus gestapelten Ereignisquellen

Wie in bereits vorangegangenen Abschnitten beschrieben, können Aktionen, die ein Prozess auslöst, zu einer Vielzahl an Unterereignissen führen, wenn für die Umsetzung ein Stapel von Softwarekomponenten zum Einsatz kommt, bei dem jede Schicht Ereignisse auslöst. Das plakative Beispiel ist die Netzwerkkommunikation durch den TCP/IP-Protokollstapel. Das Laden einer Web-Ressource (HTTP/HTTPS Ressource) als aggregierte Aktion, die ein Prozess durchführt, gliedert sich in viele Einzelereignisse. Auf der Anwendungsschicht wären das z.B. das Senden der HTTP-Anfrage und das Empfangen der HTTP-Antwort. Auf der darunterliegenden Transportschicht sind die Ereignisse für den Aufbau und Abbau der TCP-Verbindung zum Zielserver, der die Web-Ressource bereitstellt, auszumachen und zusätzliche TCP-Ereignisse, die das Versenden und Empfangen von TCP-Segmenten anzeigen. Eventuell kommen zusätzliche Ereignisse für Sende- und Empfangsbestätigungen und die Datenfluss-Kontrolle dazu. Eine weitere Ebene tiefer, auf der Vermittlungsschicht sind Ereignisse, die das Senden und Empfangen der einzelnen IP-Pakete signalisieren. Eine solche hierarchische Sequenz von mehreren Einzelereignissen kann zu einem Gesamtereignis aggregiert werden, wie es im Schaubild zu sehen ist. Anstatt einer Aggregation aller beschriebenen Ereignisse stattdessen nur Ereignisse aus der Anwendungsschicht zu berücksichtigen, erzeugt einen blinden Fleck in der Verhaltensüberwachung. Prozesse könnten ein eigenes Anwendungsprotokoll implementieren oder Bibliotheken für die Implementierung des Anwendungsprotokoll benutzen, die über keine Instrumentierung verfügen im Sinne, dass sie Event-Objekte im Kontext einer Ereignisprotokollierung erzeugen. So erscheint es für dieses Beispiel der Netzwerkkommunikation sinnvoll, für Ereignisse, die sich einem Vorgang und somit Ereignissen einer höheren Schicht zuordnen lassen, eine solche Zuordnung vorzunehmen, und diese zu einem Gesamtereignis zu aggregieren. Gleichzeitig sollten Ereignisse, für die diese Zuordnung nicht möglich ist, dennoch protokolliert und betrachtet werden.

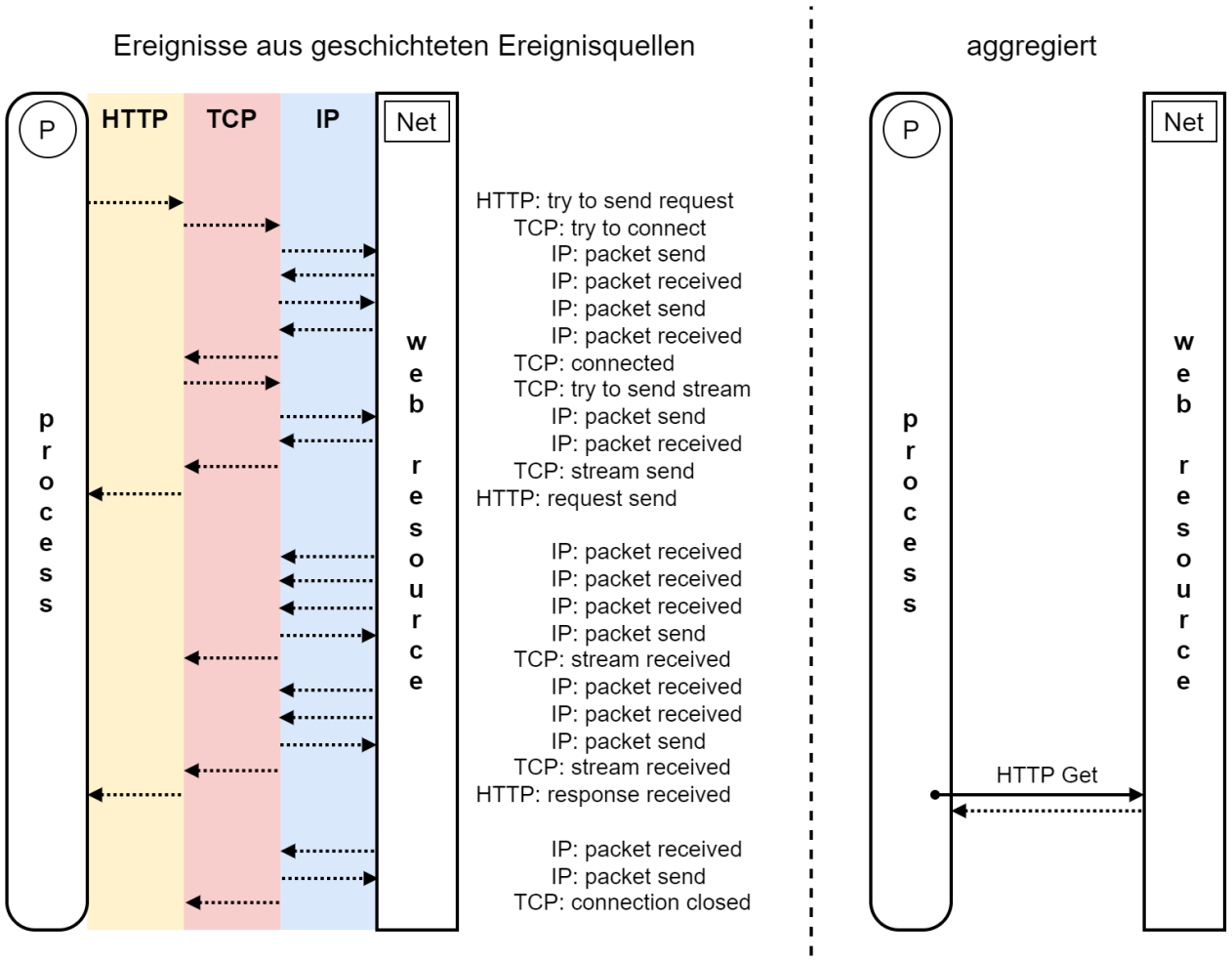

Abbildung 1.17: Gestapelte Ereignisquellen am Beispiel des Netzwerk-Protokollstapels

1.6.3 Stellvertretende Durchführung von Aktionen

In manchen Fällen führt eine Software einen Vorgang wie beispielsweise das Erstellen eines Registry-Keys nicht direkt selbst durch, sondern benutzt dazu eine andere Software als Werkzeug. Diese Werkzeuge sind in der Regel Kommandozeilenanwendungen für die Eingabeaufforderung (CMD) oder PowerShell (PS). Im Falle von Registry-Operationen kann dies z.B. das Programm `RegEdit` (`regedit.exe`) sein. Ein solches Werkzeugprogramm kann von dem Prozess, der indirekt auf die Registry zugreifen möchte, direkt als Kindprozess gestartet werden. Dem Prozess werden dabei direkt die Argumente für die eigenen Commandline-Parameter (CLI) übergeben. Der erzeugte Kindprozess führt stellvertretend die Registry-Zugriffe durch, liefert eventuell Ergebnisse als Zeichenketten über eine Unnamed-Pipe (`stdout`) dem Elternprozess zurück und beendet sich. Alternativ kann im Gegensatz dazu, dass Werkzeugprogramm

direkt als Kindprozess zu starten, auch de Eingabeaufforderung (CMD) oder die PowerShell (PS) gestartet werden, um das Werkzeugprogramm innerhalb einer Kommandozeilenumgebung auszuführen. Da die Kommunikation in diesem Fall nicht über Named-Pipes sondern einem Konsolen-Objekt läuft, hat dies unter Windows immer auch das Starten eines Konsolenhost-Prozesses zur Folge. In aktuellen Windows-Version ist dies ein `conhost`-Prozess (`conhost.exe`), der als weiterer Kindprozess des CMD-Prozesses oder PowerShell-Prozess (`powershell.exe`) gestartet wird. Dem jeweiligen Kommandozeileninterpreter (CMD oder PowerShell) wird mit der Prozesserzeugung eine oder mehrere Zeilen Skript-Code übergeben. Dies geschieht als Argument eines Startparameters (CLI). Der Kommandozeileninterpreter-Prozess führt die übergeben Kommandos aus und startet dabei das entsprechende Werkzeugprogramm als weiteren Kindprozess, der dann schlussendlich die Zugriffe stellvertretend für den initialen Prozess durchführt. Dieses beschriebene Muster ist insbesondere bei Malware häufig zu beobachten.

Zugriffe, welche durch die zwei beschriebenen Varianten indirekt über Kindprozesse legitimer Werkzeugprogramme durchgeführt werden, könnten zu einem direkten Zugriffsereignis zusammengefasst werden, welches dem initialen Prozess zugeordnet wird. Dies hat den Vorteil dass verschiedene Wege, die den selben Zweck dienen und gleiche Ergebnisse produzieren, vereinheitlicht werden und so die Weiterverarbeitung von Ereignissen vereinfachen. Zudem wird dadurch die Anzahl der Ereignisse und involvierten Subjekte und Objekte reduziert.

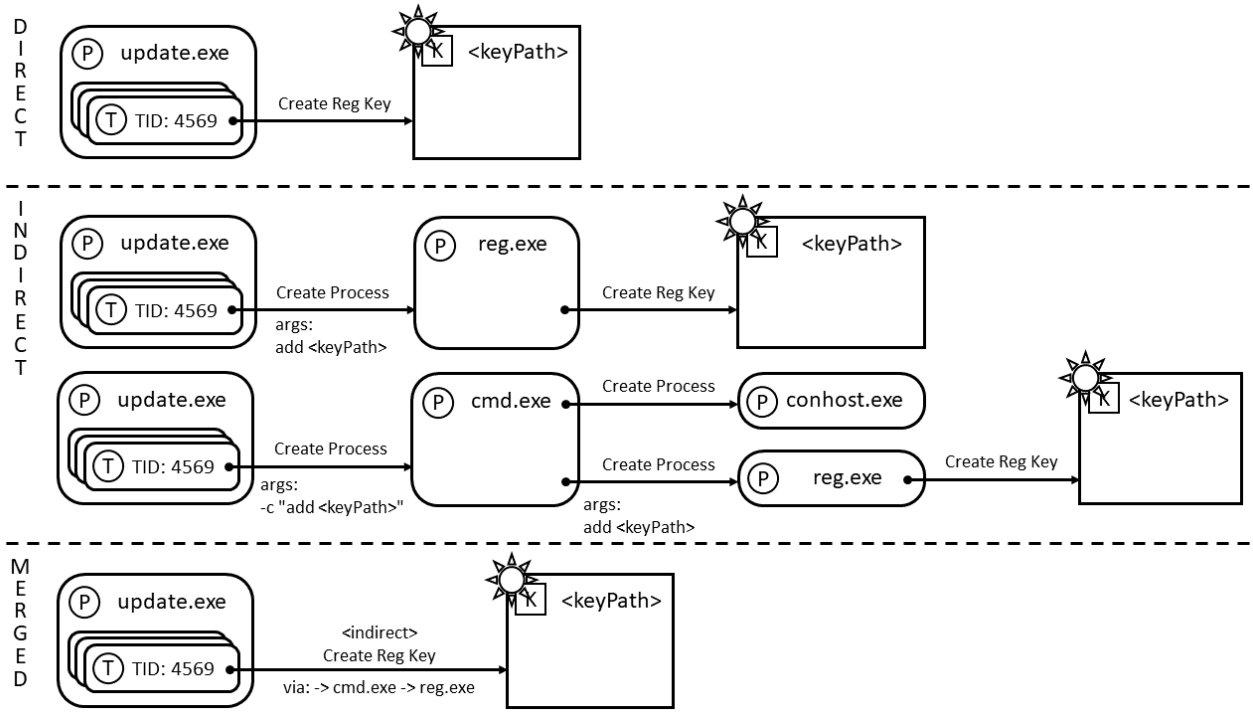

Abbildung 1.18: direkte und indirekte Durchführung von Aktionen

1.7 Sensorik

Im Folgenden sollen vier grundsätzliche Methoden aufgezeigt werden, die unter Windows benutzt werden können, um bestimmte Aktionen eines Prozesses zu detektieren und die entsprechenden Ereignisse zu protokollieren. Aktuelle IT-Sicherheitslösungen wie z.B. EDR-Sensoren nutzen meist eine Kombination der verschiedenen Verfahren, um sowohl die Erkennung wie auch eine direkte, synchrone Verhinderung bösartiger Aktionen zu gewährleisten.

1.7.1 API-Hooking

Das Windows-Betriebssystem stellt den gehosteten Anwendungen eine ausführlich dokumentierte und versionsstabile API bereit, um die Funktionalitäten des Betriebssystems zu nutzen. Diese als Windows-API (Win-API / Win32-API) bezeichnete Abstraktionsschicht existiert als eine Reihe von User-Mode-Bibliotheken (DLLs) [Wik22c], welche die Anwendungen je nach Bedarf laden und verwenden. Diese DLLs der Windows-API benutzen selbst eine zentrale Bibliothek für die Kommunikation mit dem Kernel und die dafür notwendige Durchführung der Systemcalls. Dieses zentrale Modul ist die NTDLL (`ntdll.dll`). Die NTDLL-Bibliothek stellt quasi für jeden Systemcall eine exportierte Funktion bereit, die den Systemcall technisch umsetzt. Da die Systemcall-Schnittstelle nicht versionsstabil ist und sich durch Windows-Updates ändern kann, ist es nicht vorgesehen, dass diese von Anwendungscode direkt verwendet wird. Das Systemcall-Interface ist zudem nicht offiziell von Microsoft dokumentiert. Auch für die Funktionen der NTDLL ist ebenfalls nicht vorgesehen, dass diese direkt verwendet werden, da diese ebenfalls nicht offiziell dokumentiert sind. Stattdessen sollen alle Interaktionen der Anwendung mit dem Betriebssystem über die Bibliotheken der Windows-API geschehen [Wik22f][Wik22c][YIRS17, S. 69ff]. Die DLLs, welche die Windows-API bereitstellen, sind Teil des Windows-Subsystems, der Umgebung für Windows-Anwendungen. Neben dem Windows-Subsystem existieren noch weitere Umgebungen für z.B. OS/2, POSIX und Linux (WSL: Windows Subsystem for Linux), um Anwendungen für andere Plattformen unter Windows auszuführen. Alle Subsysteme verwenden NTDLL als native API für die Kommunikation mit dem Windows-Kernel. Da auf klassischen PCs in der Regel nur das primäre Windows-Subsystem verwendet wird, beziehen sich alle Beschreibungen in dieser Arbeit auf das Windows-Subsystem. Werden 32-Bit Programme auf einer 64-Bit-Windows-Version ausgeführt, befindet sich im Bibliotheksstapel noch eine WOW-Emulationsschicht (WOW: Windows on Windows), welche die 32-Bit-API auf die 64-Bit-API abbildet.

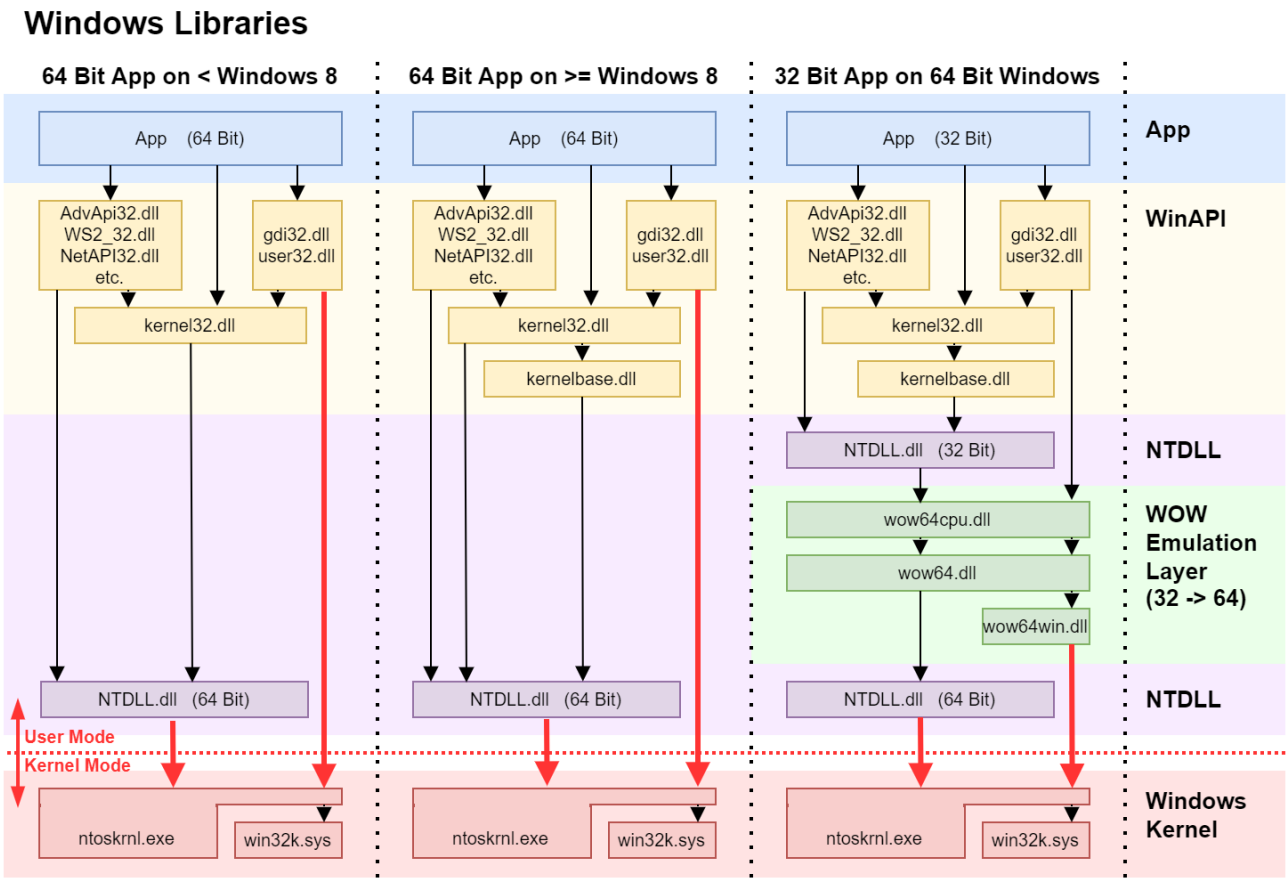

Abbildung 1.19: Windows-API

Mit der Annahme, dass jede Interaktion eines Prozesses mit externen Ressourcen wie anderen Prozessen, Dateien, Registry-Einträgen, Netzwerk-Ressourcen usw. über das Betriebssystem und spezieller über die Bibliotheken der Windows-API geschehen muss, werden diese Bibliotheken zu einem exponierten Ziel für mögliche Sensorik der Verhaltensüberwachung. Auch Malware-Code in einem Malware-Prozess oder einer anderen infizierten Anwendung, der im User-Mode ausgeführt wird, nutzt meist auch die Funktionen der Windows-API, um auf Systemressourcen zuzugreifen.

Das API-Hooking stellt eine Methode dar, die Windows-API zu instrumentieren, sodass die Zugriffe auf Systemressourcen durch den User-Mode-Code detektiert werden können. Dafür existiert meist eine Hooking-Bibliothek in Form einer weiteren DLL in den Adressräumen aller zu überwachender Prozesse. Sicherheitssoftware, ausgestattet mit den passenden Berechtigungen, ist es möglich, von außen zu veranlassen, dass eine entsprechende Hooking-DLL der Sicherheitssoftware von beliebigen Prozessen des Systems geladen wird. Beispielsweise existiert eine Registry-Konfiguration, mit der eine Liste an DLL-Bibliotheken spezifiziert werden kann, die automatisch bei der Prozesserzeugung in jeden neu erzeugten Prozess geladen werden. Die Hooking-Bibliothek installiert Hooks in den Bibliotheken der Windows-API im Adressraum des zu überwachenden Prozesses.

Dabei wird der Code der Windows-API-Funktionen leicht modifiziert. An den Anfang jeder zu „hookenden“ Funktion werden die Bytes mit einem umleitenden Sprung überschrieben, der die Ausführung in entsprechende Funktionen der Hooking-DLL umleitet. In dieser wird der Aufruf der entsprechenden API-Funktion protokolliert und der Sicherheitssoftware signalisiert. Je nach Regeln und Weiterverarbeitung kann daraus das entsprechende Ereignis detektiert werden und bei Erkennung bösartigen Verhaltens der Aufruf der Windows-API-Funktion durch den Anwendungscode unterbunden werden. Am Ende der Verarbeitung innerhalb der Hooking-DLL wird hinter den installierten Hook zurückgesprungen, sodass die API-Funktion korrekt weiter ausgeführt werden kann. Eventuell müssen durch den Hook zerstörte Befehle noch vorher durchgeführt werden, sodass der Maschinenstatus mit dem übereinstimmt, der bei einer normalen Ausführung ohne Hook existiert hätte. Letzteres gestaltet sich unter Windows recht problemlos, da die Windows-Bibliotheken das Hot-Patching-Feature unterstützen, welches Sicherheitslösungen, Debuggern, Profiling und Monitoring-Software besonders gute Bedingungen für die Installation von Hooks schafft. Dabei beginnt jede exportierte Funktion mit einer Folge aus NOP-Befehlen. Diese können mit dem Hook-Sprung-Code überschrieben werden, ohne das wichtige Instruktionen dabei zerstört werden [Mos21][Bau21] [Ste20].

Um die Überwachung durch API-Hooking zu umgehen, versuchen Malware-Autoren Code zu schreiben, der in die Funktion nicht korrekt beim Funktionsbeginn einspringt, sondern einige Befehle dahinter und somit hinter den umleitenden Sprung. Eventuell müssen Sie dabei auch selbst die Wirkung übersprungener Befehle korrigieren [Mos21].

Eine andere Maßnahme dem API-Hooking in den Windows-API-Bibliotheken zu umgehen, stellt die direkte Nutzung der exportierten Funktionen der NTDLL dar. Die NTDLL-Schnittstelle ist kleinteiliger, technischer und nicht besonders komfortabel zu benutzen. Zudem ist sie offiziell nicht dokumentiert, da Microsoft es nicht vor sieht, dass diese direkt verwendet wird. Jedoch ist diese Schnittstelle im Vergleich zur Systemcall-Schnittstelle weitestgehend versionsstabil. Aus diesem Grund benutzt Schadsoftware in einigen Fällen direkt die Funktionen der NTDLL. [Mos21].

Um diesem Problem entgegenzuwirken, versucht Sicherheitssoftware in gleicher Weise die exportierten Funktionen der NTDLL zu hooken. Auch hier versuchen die Malware-Autoren die Hooks durch Überspringen oder andere Methoden zu umgehen, wohingegen die Gegenseite versucht, die Hooking-Methoden robuster zu machen. Das skizzierte Beispiel stellt dabei nur eine einfache Methode des Hookings dar. Darüber hinaus existieren weitere, weitaus komplexere Methoden, die in produktiven Sicherheitslösungen Verwendung finden, hier aber nicht weiter vertieft werden sollen [Mos21].

Ein noch weitreichenderer Ansatz jegliche API-Hookings zu umgehen, stellt die Möglichkeit dar, direkt Systemcalls auf den Windows-Kernel durchzuführen, ohne eine Windows-Bibliothek dafür zu benutzen. Ein solches Vorgehen ist stark, bezüglich des Umgehens

jeglicher auf API-Hooking basierender Sensorik [Gav18][Mos21]. Jedoch birgt dieses Vorgehen ein großes Problem für den Malware-Autor. Abgesehen davon, dass die Systemcall-Schnittstelle technisch kompliziert zu benutzen und von Microsoft nicht offiziell dokumentiert ist, ist sie auch nicht versionsstabil. Eine Malware, die von solchen Methoden Gebrauch macht, ist dadurch an eine Reihe von konkreten Build-Versionen des Betriebssystems gebunden. Befindet sich auf dem anzugreifenden Zielsystem eine falsche Build-Version von Windows, würde die Malware nicht korrekt funktionieren. Zudem könnte ein auf dem Ziel-System durchgeführtes Windows-Update potentiell das Systemcall-Interface ändern und somit die Funktionalität der Malware beeinträchtigen. Dies stellt ein großes Risiko für den Einsatz einer solchen Malware in einer länger andauernden Angriffskampagne dar. Ist das Zielsystem genau bekannt und sichergestellt, dass dieses in der Regel nicht aktualisiert wird, wäre ein solcher Ansatz aber denkbar. Letzteres Szenario wäre vielleicht im Umfeld von Windows auf Embedded-Systems denkbar. Zusammenfassend ist zu sagen, dass die direkte Nutzung der Systemcall-Schnittstelle durch Malware sehr selten ist [Gav18].

1.7.2 Kernel-Callbacks

Zu dem im vorherigen Abschnitt beschriebenen Umgehen des API-Hookings ist Malware auch prinzipiell in der Lage, das Hooking zu manipulieren. Die Schadsoftware könnte in gleicherweise, wie der Code in der Hooking-DLL die API-Funktionen patcht, die Hooks ebenfalls durch Überschreiben zu zerstören versuchen. Gegebenenfalls wäre es auch möglich, das geladene Image der Hooking-DLL selbst zu verändern und deren Funktionalität dadurch zu stören oder zu manipulieren.

Der Grund, dass Malware das API-Hooking im User-Space umgeht oder manipuliert, zwingen die Sicherheitsfirmen dazu, die Sensorik in den Kernel zu verlagern, wo diese sich sicher außerhalb des Aktionsradius eines im User-Mode ausgeführten Malware-Codes befindet. Hierfür muss die Sicherheitslösung einen eigenen Kernel-Mode Treiber in den Windows-Kernel einbringen. Ein solches Kernel-Modul von Drittanbietern muss den aufwendigen und teuren Zertifizierungsprozess durchlaufen, um mit einer Microsoft Signatur ausgestattet, auf Produktivsystemen der Kunden in den Kernel geladen werden zu können [Yos20, S. 345ff][YIRS17, S. 661ff]. Ein Kernel-Modul bietet einer Sicherheitslösung jedoch gute Möglichkeiten, eine Sensorik zur Verhaltensüberwachung von Prozessen zu implementieren.

Der Windows-Kernel bietet für Kernel-Module die Funktionalität Callback-Funktionen für eine große Menge an Vorgängen bzw. Ereignissen innerhalb des Windows-Kernels registrieren [Yos20, S. 193ff][Yos20, S. 211ff][Yos20, S. 214ff][Yos20, S. 233ff][Yos20,

S. 217ff]. Führt der Kernel eine bestimmt Aktion durch, für die er eine Callback-Funktionalität anbietet, wird aus dem Kernel-Code direkt in die verschiedenen für dieses Ereignis registrierten Callback-Routinen synchron eingesprungen. In den Callback-Routinen können die Treiber auf das entsprechende, eingetretene Ereignis reagieren und es behandeln. Für einen Ereignistyp können verschiedene Module parallel Callbacks-Funktionen registriert haben. Die Einsprungadressen der registrierten Callback-Funktionen werden kernelseitig in einer Array-Liste verwaltet [Bau21][Ste20]. Die Callback-Funktionen werden dabei in der Reihenfolge, in der sie registriert wurden, nacheinander synchron aufgerufen. Windows bietet für die meisten Vorgänge zwei Ereignisse an, für die Callbacks registriert werden können. Ein Ereignis, welches ausgelöst wird, bevor die Aktion durchgeführt wird (Pre-Ereignis), und ein Ereignis, das ausgelöst wird, nachdem die Aktion durchgeführt wurde (Post-Ereignis). Dabei wird das Pre-Ereignis durch die Windows-API signalisiert, die einen speziellen Systemcall tätigt, der den Kontrollfluss in den Kernel verzweigt und dort zum Aufruf der Pre-Callbacks führt. Dass die Aktion nicht aus dem Kernel-Code sondern aus dem Code der API-Bibliotheken heraus stattfindet, hat zur Folge, dass nach der Signalisierung noch der eigentliche Aufruf auf die NTDLL durchgeführt wird. Dies hat den Vorteil, dass eine Sicherheitslösung diesen Aufruf der NTDLL auf Anfrage hooken könnte und somit die Software daran hindern kann, die Aktion durchzuführen [Bau21][Ste20].

Ein Sicherheitstreiber registriert für eine Reihe relevanter Ereignisse, die er überwachen möchte, wie z.B. die Erstellung von Kindprozessen oder das Laden von Modulen, Callback-Funktionen für die entsprechenden Pre- oder Post-Ereignisse. Führt die zu überwachende Anwendung eine entsprechende Aktion durch, wird das Pre-Ereignis über einen Systemcall dem Kernel signalisiert. Das Post-Ereignis tritt ein, wenn der Kernel-Code für die entsprechende Aktion durchlaufen ist. Der Kernel-Code ruft in beiden Fällen die Callback-Funktionen synchron im Kontext des Threads und Prozesses, der die Aktion durchführen möchte oder durchgeführt hat, auf [Yos20, S. 193ff][Yos20, S. 217ff]. Der Sicherheitstreiber kann den Vorgang auswerten, protokollieren und darauf reagieren. In der Regel wird das Ereignis über die Geräte-Objekt-Schnittstelle oder eine andere Treiber-Schnittstelle einem im User-Mode laufenden Sicherheitsdienst-Prozess signalisiert, der die weitere Verarbeitung durchführt. Soll auf die Aktion reagiert werden bevor diese durchgeführt wird oder diese sogar verhindert werden, darf die Callback-Funktion nicht beendet werden, da sonst der Code im Kernel weiter ausgeführt wird und die Aktion stattfindet. Der Sicherheitstreiber besitzt dazu die Möglichkeit den Thread an dieser Stelle innerhalb der Callback-Funktion zu blockieren (schlafen zu legen). Dadurch besteht für die Sicherheitssoftware und deren parallel laufende Prozesse das Zeitfenster, um das Ereignis zu verarbeiten und eine Reaktion zu implementieren [Bau21][Ste20].

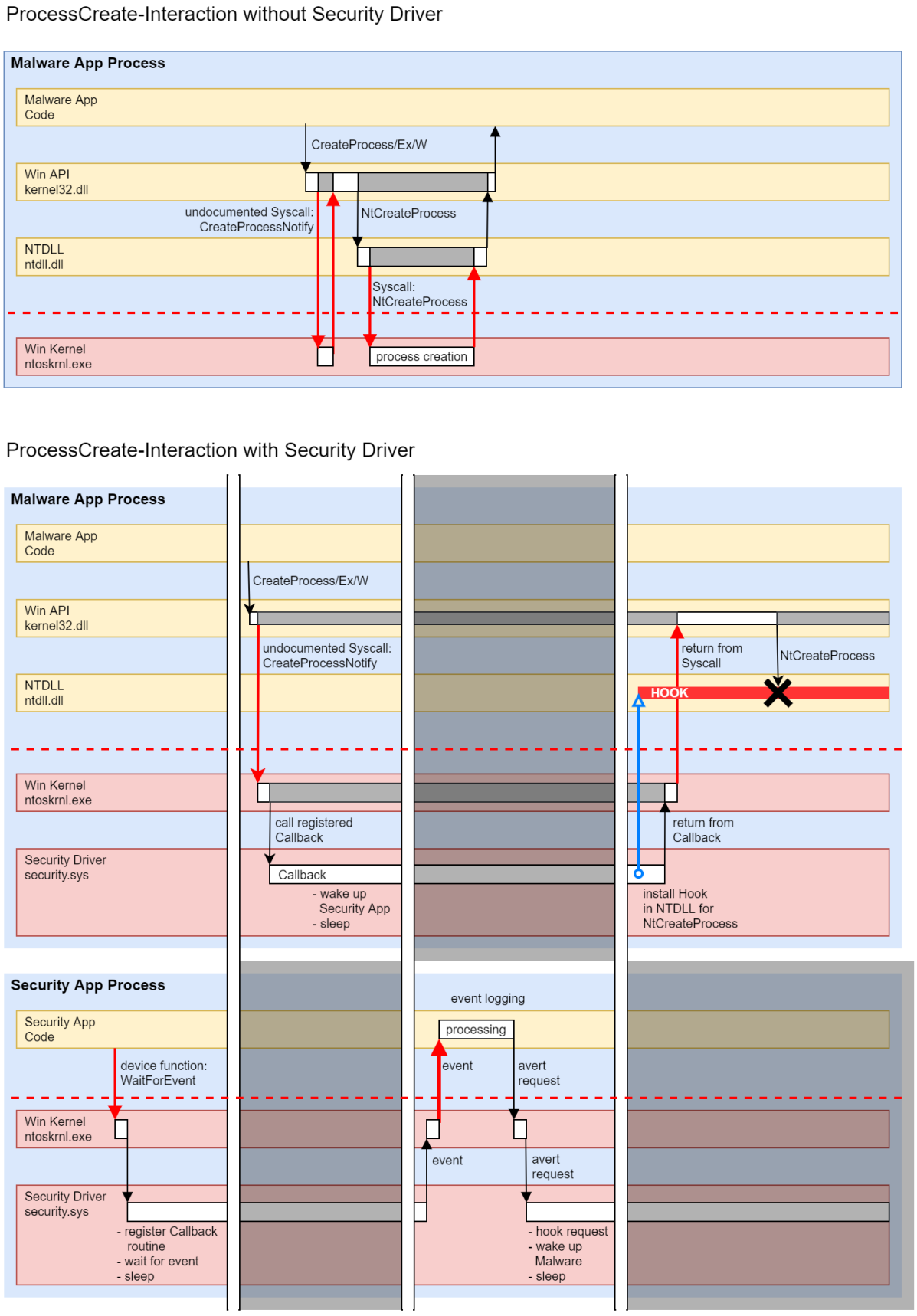

Abbildung 1.20: API-Hooking als Reaktion auf ein via Kernel-Callback erkanntes Ereignis [Bau21][Ste20]

Eine mögliche Implementierung könnte sein, dass die entsprechende Funktion der NTDLL, die nach der Signalisierung des Pre-Ereignisses aufgerufen werden würde, via API-Hooking manipuliert wird, sodass der Vorgang nicht durchgeführt werden kann. Dieser Hooking-Vorgang kann durch den Sicherheitstreiber oder durch eine in den Prozess injizierte Sicherheits-Bibliothek durchgeführt werden. Ist dies passiert wird der Thread wieder durch den Sicherheitstreiber aufgeweckt. Dadurch wechselt sein Zustand zu bereit, sodass er an der Stelle innerhalb der Callback weiter ausgeführt werden kann [Bau21][Ste20].

Seltenere, komplexere Malware besitzt manchmal die Möglichkeiten, den Kernel des Betriebssystems zu infizieren oder zu manipulieren. Dabei werden die Schnittstellen von Kernel oder Gerätetreibern exploitiert, um den Kernel zu veranlassen bestimmte Dinge zu tun. Dies könnte z.B. die Ausweitung der eigenen Berechtigungen oder das Ausführen von Code im Kernel-Mode sein. Eher seltener bringt die Malware dafür einen eigenen Kernel-Mode-Treiber in den Kernel ein. Häufiger veranlasst die Malware das Laden legitimer Gerätetreiber mit Sicherheitslücken um diese zu exploitieren. Gelingt es einer Malware den Kernel zu infizieren, ist es ihr prinzipiell möglich, den Sicherheitstreiber zu Stören oder die Callback-Tabellen des Kernels so zu manipulieren, dass die installierten Callbacks nicht mehr aufgerufen werden [Bau21][Ste20][Mos21].

1.7.3 Filtertreiber

Nicht für alle Aktionen existiert eine Callback-Funktionalität. Manche Vorgänge, in die weitere Treiber wie z.B. Dateisystemtreiber oder Gerätetreiber wie Netzwerktreiber oder die Treiber von Datenträgern involviert sind, können über eine andere Art detektiert werden. Dabei muss die Sicherheitssoftware wiederum ein Kernel-Modul einbringen, welches in gleicher Weise von Microsoft signiert sein muss, damit es auf Produktivsystemen verwendet werden kann. Diesmal handelt es sich um einen Filtertreiber [LB19, S. 24ff]. Berühren Aktionen I/O-Geräte (echte oder virtuelle Software-Geräte) wie Dateizugriffe, Netzwerkzugriffe usw. werden diese Vorgänge vom IO-Manager des Kernels koordiniert. Dabei durchläuft die Zugriffs-Anfrage des Prozesses die Schichten des Stapels aus Geräte-Objekten (Gerätestapel), der sich aus dem Gerätebaum ableitet [Yos20, S. 166ff][YIRS17, S. 649ff][YIRS17, S. 613ff]. Mit den Geräte-Objekten sind Treiber assoziiert, welche die Anfragen behandeln und dann an die darunter liegende Schicht im Stapel weiterleiten. An bestimmten Stellen im Gerätestapel ist es Treibern möglich, Filtergeräte in den Stapel einzufügen. Die Anfragen müssen dadurch zuerst von den dahinter liegenden Filtertreibern behandelt werden, bevor sie an das Geräte bzw. den dahinter liegenden Treiber weitergereicht werden. Die Filtertreiber erhalten stattdessen die Anfrage und reichen sie nach unten zum nächsten Filtertreiber oder dem Gerätetreiber weiter durch. Der Filtertreiber sieht dabei alle Daten, die für die Anfrage von einer Schicht im Gerätestapel

zur nächsten darunterliegenden Schicht notwendig sind [LB19, S. 8ff]. Diese Anfragedaten sind als IO-Request-Paket (IRP) strukturiert [Yos20, S. 159ff][YIRS17, S. 590ff]. Es ist möglich aus den Daten des IO-Request-Paketes (IRP) die Vorgänge wie z.B. Zugriffe auf Dateien, die Registry, das Netzwerk und andere Ressourcen zu erkennen, diese zu protokollieren und darauf zu reagieren [LB19, S. 10ff][LB19, S. 24ff][LB19, S. 25ff]. Die Reaktion könnte in der Weise erfolgen, dass beispielsweise der Transfer der Anfragedaten in eine darunterliegende Treiberschicht nicht durchgeführt wird oder die Anfragedaten manipuliert werden [Yos20, S. 243ff]. Der Sicherheits-Filtertreiber kommuniziert dazu ebenfalls über eine Schnittstelle mit einem User-Mode-Dienst-Prozess der Sicherheitsanwendung. Gegebenenfalls muss dazu der zu überwachende Thread innerhalb des Codes des Filtertreibers blockiert (schlafen gelegt) werden. Dies entspricht in etwa dem Vorgehen, wie es im vorherigen Abschnitt über die Callbacks beschrieben ist. Der Code des Filtertreibers wird dabei zwei Mal durchlaufen, wenn das IRP-Paket nach unten gegeben wird, also vor der Durchführung des zu überwachenden Vorgangs, und nach der Durchführung des zu überwachenden Vorgangs, wenn das IRP-Paket wieder nach oben der Instanz zurückgegeben wird, welche die Anfrage gestellt hat. Filtertreiber bieten damit interessante Möglichkeiten, auf erkannte Ereignisse zu reagieren, indem beispielsweise ein Ereignis nicht durchgeführt wird, der anfragenden Instanz jedoch eine Antwort mit entsprechenden Fake-Daten zugestellt wird, die dem Anfrager suggeriert, das der Vorgang erfolgreich durchgeführt wurde [Yos20, S. 243ff]. Somit besteht die Möglichkeit, User-Mode-Software effektiv zu täuschen. Diese Täuschung, die einer Sicherheitssoftware oder Analysesoftware nützlich sein kann, um das Malware-Verhalten zu erkennen, wird im Gegenzug durch Kernel-Mode-Malware (Rootkit) ausgenutzt. Schadsoftware kann über eigene Filtertreiber oder das Exploitieren und anschließende Manipulieren eines Filtertreibers, die Sichtbarkeit jeglicher Prozesse und Systemressourcen einschränken. Dies ermöglicht Malware sich besser zu verstecken. Solche Funktionalitäten werden aktiv von Rootkits benutzt.

Windows bietet zwei Varianten an, Filtertreiber zu implementieren. Die ältere Variante wird mittlerweile als Classic-Filter-Driver bezeichnet. Dabei installieren die Filter-Treiber ähnlich normaler Gerätetreiber Filtergeräte im Stapel [LB19, S. 10ff][LB19, S. 24ff][LB19, S. 25ff]. Diese Treiber können nicht im laufenden Betrieb geladen oder entladen werden, für beides ist ein Neustart des Systems erforderlich. Die neuere Variante wird als Mini-Filter-Driver bezeichnet. Mini-Filter werden über einen Filtermanager verwaltet und können ohne Neustart im laufenden Betrieb geladen und entladen werden [Yos20, S. 361ff]. Der Filtermanager bietet Treibern die Möglichkeit Callbacks für die Überwachung von Geräte-Objekten zu registrieren. Diese werden aufgerufen, bevor IRP-Pakete als Anfrage von einen Treiber empfangen werden oder gesendet werden [LB19, S. 25ff].

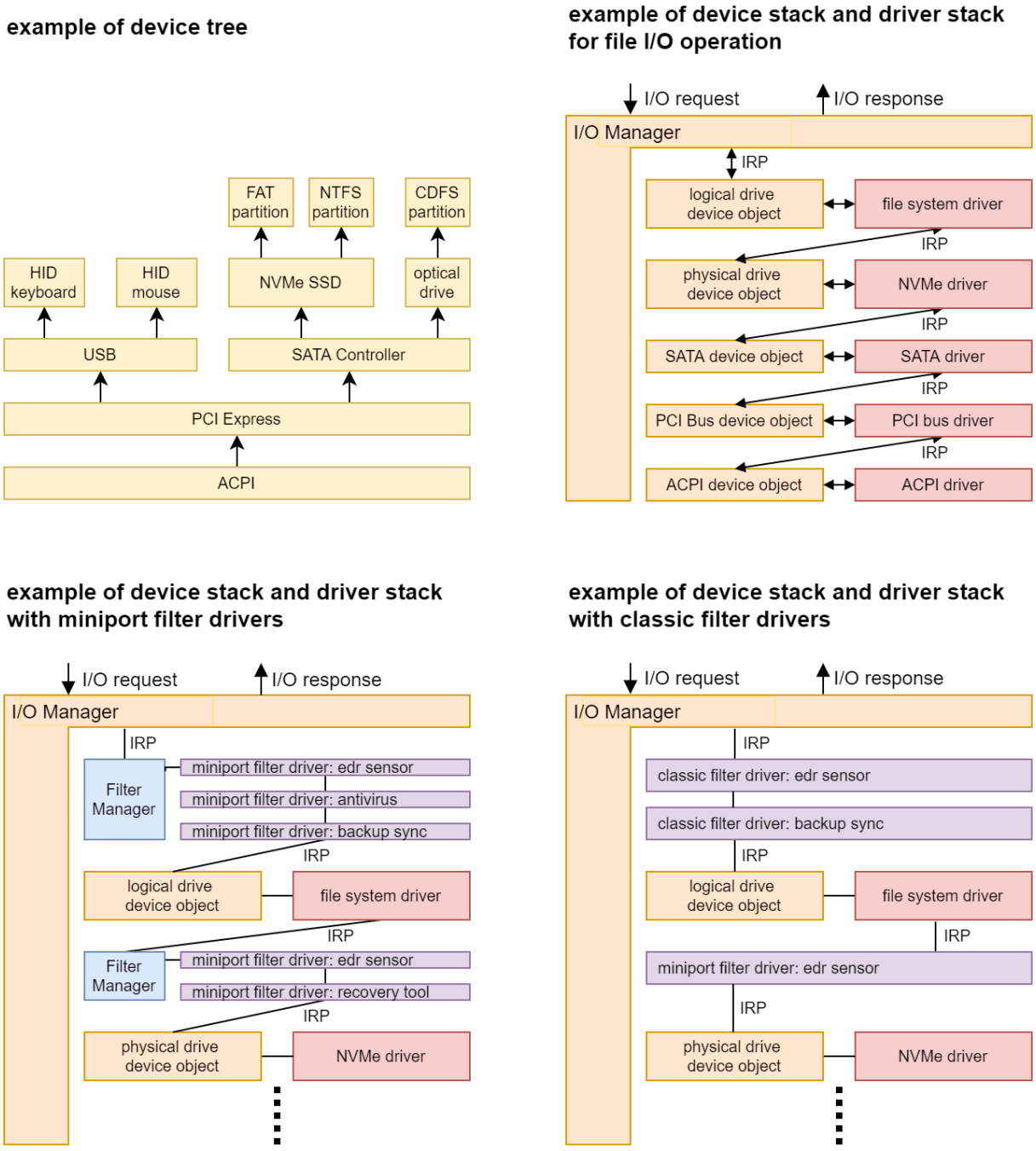

Abbildung 1.21: Filtertreiber und Filtermanager im Gerätestapel

1.7.4 Event Tracing for Windows (ETW)

Event-Tracing-for-Windows (ETW) ist eine Funktionalität des Betriebssystem-Kernels, die es User-Mode-Prozessen erlaubt, Event-Objekte zu erhalten, die Ereignisse, die auf dem System aufgetreten sind, signalisieren. Event-Objekte können dabei von verschiedensten Instanzen generiert werden. Das können sowohl Komponenten im Windows-Kernel sein, wie auch User-Mode-Komponenten. Anwendungen, Services und Systemprozesse oder deren Bibliotheken können in gleicher Weise ETW-Events erzeugen, wie der Kernel-Code des Windows-Kernels oder geladener Treiber. Viele Windows-eigene Komponenten erzeugen Event-Objekte, darunter viele Teile des Kernels, der User-Mode Bibliotheken, der Systemprozesse und internen Werkzeuge und Windows-Dienste. ETW ermöglicht es auch Drittanbieter-Software Event-Objekte für ETW zu erzeugen und diese darüber, Monitoring-Software zur Verfügung zu stellen. Die Event-Objekte werden durch ETW vom Erzeuger zu den konsumierenden Analyse-Anwendungen transportiert. Wobei die Monitoring-Software spezifizieren kann, welche Ereignisse sie erhalten möchte [AIRS21, S. 499ff][Yos19, min. 1:26][Gol17, min. 4:15].

Sicherheitssoftware wie EDR-Sensoren setzen ETW stark als Sensor-Quelle ein [TKG21b, min. 5:47]. Klassisches ETW hat dabei den Vorteil, dass die Menge an erzeugenden Komponenten sehr groß ist und detailliert alle Abläufe auf der Systemebene umfasst. Mit ETW kann somit das Zugriffsverhalten von zu überwachenden Prozessen gut gemessen werden [TKG21b, min. 5:19]. Zudem ist dies von einem externen Prozess, vom User-Mode aus, möglich. Für das Beziehen von ETW-Ereignissen ist kein Kernel-Modul notwendig, was potentiell die Systemstabilität erhöht, die Komplexität der Sicherheitssoftware verringert und die Microsoft-Zertifizierung des Treiber-Codes überflüssig macht [Rot21, min. 3:30][Rot22b, min. 4:00]. Klassisches ETW bringt jedoch im Vergleich zu den zuvor vorgestellten Methoden einen starken Nachteil mit sich. Zwischen des Erzeugens eines Event-Objektes und dem Eingehen dieses bei einer konsumierenden Analyse-App vergeht eine Zeitspanne. Diese kann in ungünstigen Fällen bis zu ca. drei Sekunden betragen [Yos19, min. 26:00] Meist ist diese aber kleiner, jedoch groß genug, dass eine erhebliche Menge an Befehlen in der Zwischenzeit ausgeführt werden kann. Zudem bietet klassisches ETW von sich aus keine Möglichkeiten auf den ereignisauslösenden Vorgang direkt (synchron) zu reagieren oder diesen zu verhindern. Die Signalisierung des Ereignisses erfolgt bei ETW immer asynchron. Dadurch sind, nachdem die Sicherheitssoftware über das Ereignis informiert wurde, auf einem modernen Computersystem die meisten Zugriffsvorgänge lange abgeschlossen. Aus diesem Grund benutzt Sicherheitssoftware ETW meist nur als zusätzliche Sensorkomponente neben den, in den vorherigen Abschnitten aufgeführten, Verfahren. Bisher kann für einen synchronen Eingriff in den Vorgang durch die Sicherheitssoftware nicht auf Kernel-Callbacks und Filter-Funktionalitäten des IO-Systems verzichtet werden (abgesehen von DTrace).

In neueren Versionen von Windows 10 existiert DTrace, eine zusätzliche Instrumentierungsmethode für den Windows-Kernel. DTrace ist ein ursprünglich von Sun Microsystems für das Betriebssystem Solaris (UNIX-Derivat) entwickeltes Kernel-Instrumentierungswerkzeug. DTrace wurde auf verschiedene UNIX- und Linux-Betriebssysteme portiert. Seit dem 19H1-Update von Windows 10 (Mai 2019) steht DTrace auch standardmäßig im Windows-Kernel bereit. Mit DTrace kann bei beliebigen Funktionsaufrufen im Windows-Kernel oder beim Durchlaufen bestimmter gesetzter Breakpoints synchron Code-Sequenzen (Instrumentierungssequenzen) ausgeführt werden, die von einer externen Instrumentierungs-Software spezifiziert wurden. DTrace für Windows bietet zudem die Möglichkeit diese Instrumentierungssequenzen auch beim Auftreten bestimmter ETW-Ereignisse synchron durchführen zu lassen. Dies würde Sicherheitssoftware ermöglichen, alle durch ETW-Ereignisse signalisierten Vorgänge in gleicher Weise wie Kernel-Callbacks zu unterbrechen. Die Instrumentierung mit DTrace soll im Produktivbetrieb mit aktiviertem PatchGuard (KPP) und HyperGuard (VBS) möglich sein [AIRS21, S. 525ff].

2 Auftrittsfrequenzen von Ereignissen

Eine praktische Fragestellung, die sich im Kontext des Protokollierens, der weiterverarbeiten und der direkten Reaktion auf Ereignisse ergibt, zielt auf den zusätzlichen Aufwand für die Erzeugung und Verarbeitung der Eventdaten und deren Auswirkung auf das zu überwachende System. Da die Sensorik zur Überwachung und eventuell Verarbeitungskomponenten auf dem selben Rechner ausgeführt werden, der überwacht wird, konkurrieren diese Komponenten mit der zu überwachenden Software um die Systemressourcen. Die mit unter meist entscheidende und limitierende Größe ist dabei die Rechenzeit, die zusätzlich für das Monitoring benötigt wird. Daraus resultiert eine Verlangsamung des Gesamtverhaltens des Computersystems. In Fällen einer Überwachung eines interaktiven Systems, an dem Endbenutzer arbeiten, sollte diese möglichst klein ausfallen. Im Idealfall sollte die Verlangsamung so gering sein, dass der Endanwender keinen oder nur einen sehr geringen und damit noch gut vertretbaren Unterschied wahrnimmt im Vergleich zu einem nicht überwachten System.

Die Gesamtlast der Event-Daten-Erzeugung und -Verarbeitung ist abhängig von der Menge zu messender Ereignisse pro Zeitintervall. Bei einer praktischen Implementierung einer Sensorik und Vorverarbeitung muss entschieden werden, welche Ereignistypen gemessen, protokolliert und weiterverarbeitet werden sollten und für welche Ereignistypen dies in welchem Detailgrad passieren soll. Alle diese Spezifika sind unter anderem vor allem von den Auftrittsfrequenzen der verschiedenen Ereignistypen abhängig. Die Auftrittsfrequenz ist somit eine wichtige Charakteristik von Ereignistypen.

Um eine Bessere Einschätzung hinsichtlich der Ereignismengen zu bekommen, soll dieses Kapitel einen groben Überblick über die Auftrittsfrequenzen bestimmter Basisereignisse eines Windows-PCs liefern. Die Auftrittsfrequenzen wurden dazu im Rahmen dieser Arbeit durch praktische Messungen ermittelt. Für die Messungen wurde Performance-Counters-for-Windows (PCW) eingesetzt.

2.1 Performance Counters for Windows (PCW)

Performance-Counters-for-Windows (PCW) ist ein im Betriebssystem-Kernel verankerter Dienst zur Leistungsüberwachung auf Basis von Zahlenwerten. Er ermöglicht Anwendungen einzelne Zahlenwerte, die als Leistungsindikatoren dienen und auch so bezeichnet werden, abzufragen. Auf der Gegenseite können Software-Komponenten wie Anwendungen, Bibliotheken und Kernel-Module dem Performance-Counter-System anbieten, auf Anfrage Zahlenwerte als Leistungsindikatoren bereitzustellen. Da die meisten Werte eine Anzahl oder Rate von etwas beschreiben, das gezählt wird, werden die Werte auch

als Leistungszähler angesehen, woraus sich der Begriff Performance-Counter ergibt. Der Windows-Kernel sowie die Windows-eigenen User-Mode-Komponenten wie Bibliotheken, Systemprozesse und Serviceprozesse stellen über diese Methode eine große Anzahl an Leistungsindikator-Werte bereit. Zudem ist es Dritt-Software möglich, über die Performace-Counter-API Leistungsindikator-Informationen bereitzustellen. Dies gilt sowohl für User-Mode Anwendungsprozesse wie auch für Kernel-Mode Treiber von dritter Stelle [Yos19, min. 9:20][YIRS17, S. 40ff][Mic3a].

Die Zielsetzung von Performance-Counters-for-Windows (PCW) liegt in der Überwachung der Systemleistung und der Leistung einzelner Softwarekomponenten auf Basis von Zahlenwerten. Dieser Zweck unterscheidet sich ausdrücklich von der Ablaufverfolgung (Tracing). PCW eignet sich nicht für die Ablaufverfolgung, da es nicht möglich ist, über die Zähler indirekt einzelne Ereignisse zu erkennen, direkt nachdem diese aufgetreten sind. Für solche Zwecke sollte Event-Tracing-for-Windows (ETW) anstelle von PCW verwendet werden. Grund dafür ist, dass das Performance-Counter-System intern die Werte maximal einmal pro Sekunde aktualisieren kann. Jeder Wert ist dabei intern immer eine vorzeichenlose Ganzzahl (unsigned integer) mit einer Breite von Wahlweise 32-Bit oder 64-Bit, die jedoch verschieden interpretiert werden kann (Anzahl, Rate/Frequenz, Prozentwert usw.). Die Leistungsindikator-Werte der Windows-Performance-Counter werden von Windows internen Werkzeugen genutzt. Dazu gehören der Task-Manager (`taskman.exe`), der Ressourcenmonitor, der Leistungsmonitor (`perfmon.exe`), `typeperf.exe`, `logman.exe` und `relog.exe` [Mic3a][YIRS17, S. 40ff].

Die Verwaltungsstrukturen von PCW sind hierarchisch aufgebaut. Im Kernel besteht ein globaler Pool an registrierten Countersets. Ein Counterset stellt eine Gruppe verschiedener Leistungsindikatoren beliebiger Anzahl dar, die einen semantischen Bezug zueinander haben und von derselben Code-Quelle stammen. Zusätzlich zu den Leistungsindikatoren kann ein Counterset auch eine beliebige Anzahl an Instanzen spezifizieren. Eine Instanz repräsentiert dabei eine Entität im System, auf die sich die Leistungsindikatoren beziehen. Es gibt auch Countersets ohne unterschiedliche Instanzen. Bei diesen sind die Leistungsindikatoren systemweit anzusehen. Instanzen könnten z.B. die verschiedenen Rechenkerne der CPU sein, die Prozesse, Threads oder Systemressourcen wie z.B. logische Datenträger, IPC-Objekte oder I/O-Geräte. Meist existiert noch eine weitere Pseudoinstanz, welche die Summe über alle Entitäten angibt. Jeder Counterset hat einen systemweit eindeutigen Namen. Ein Leistungsindikator besitzt wiederum einen eindeutigen Namen als Bezeichner. Zudem kann ein Beschreibungstext festgelegt werden, welcher die semantische Bedeutung der Werte des Leistungsindikators erklärt. Der Name muss dabei aber nur innerhalb des Countersets eindeutig sein. Beispiele für Leistungsindikatoren können z.B. die Anzahl der Systemaufrufe, Interrupts, Kontextwechsel pro Sekunde oder z.B. die Anzahl an Prozessen, Threads oder Handles sein. Auch prozentuale Werte

wie Prozessorauslastung, Interrupt-Zeitverhältnis usw. können als Leistungsindikatoren in verschiedenen Countersets bereitstehen. Wie erwähnt können Leistungsindikatoren jedoch auch mehrere Werte pro Indikator besitzen, wenn das Counterset verschiedene Instanzen enthält. Leistungsindikatoren und Instanzen bilden eine zweidimensionale Matrix, deren Zellen die eigentlichen Leistungsindikator-Werte sind. Im Bezug auf die genannten Beispiele könnte es z.B. verschiedene Instanzen für verschiedene Rechenkerne, Threads und Prozesse geben, auf die sich die Werte beziehen, sowie eine Instanz die den summierten Gesamtwert liefert. Auch die Leistungsindikator-Instanzen besitzen eindeutige textuelle Bezeichner pro Counterset. Die in einem Counterset definierten Instanzen gelten immer für alle Leistungsindikatoren des Countersets. Soll es zusätzliche Leistungsindikatoren geben, die sich auf eine andere Menge an Instanzen beziehen oder einen systemweiten Bezug ohne Instanzen aufweisen, müssen dafür separate Countersets angelegt werden [Mic3a].

PCW sieht für Software-Komponenten, die Performance-Counter nutzen möchten, die zwei Rollen Provider und Consumer vor. Provider-Komponenten registrieren dabei ein oder mehrere Countersets. Der Provider definiert die Leistungsindikatoren und gegebenenfalls Instanzen für ein Counterset, dass er beim PCW-System registrieren möchte, über ein XML-Manifest (.man). Consumer-Komponenten können mehrere verschiedene registrierte Countersets abonnieren. Dies ermöglicht ihnen in einem regelmäßigen Intervall alle Indikatorwerte jedes abonnierten Countersets zu beziehen. Das Intervall kann dabei einmal pro Sekunde oder einmal pro Minute betragen. Der dem Consumer bei jedem Intervall gelieferte Datensatz wird als Probe bezeichnet. Eine PCW-Probe beinhaltet neben den Werten der Counterset-Indikatoren für die verschiedenen Instanzen auch einen Zeitstempel, der angibt, auf welchen Zeitpunkt und indirekt auf welchen Zeitraum sich die gelieferten Werte der Probe beziehen [Mic3a].

Jede Sekunde oder jede Minute fordert der Kernel-Code vom PCW-System (`PCW.sys`) jede Provider-Komponente, dessen Counterset von mindestens einer Consumer-Komponente abonniert wurde, auf, die Werte für das Counterset bereitzustellen. So findet das eigentliche Zählen von Ereignissen oder Elementen über interne Variablen innerhalb der Provider-Komponenten statt. Jede Sekunde übermitteln diese auf Aufforderung die Werte aus ihren internen Zählvariablen an das PCW-System im Kernel, dass diese an die Consumer, die das Counterset abonniert haben, weiterleitet. Ist das Counterset von keinem Consumer abonniert, wird der Provider nicht aufgefordert und übermittelt keine Werte [Mic3a].

Es existieren verschiedene Typen von Indikator-Werten. Manche Rohwerte liefern bereits als einzelner Wert ohne Weiterverarbeitung eine nutzbringende Information. Andere Rohwerte müssen weiterverarbeitet und über mehrere Proben aggregiert werden, um eine nutzbringende Information zu generieren. Den Typ der Weiterverarbeitung, den der

Consumer durchführen muss, ist über ein weiteres Feld für jeden Indikator des Countersets in der Probe hinterlegt. Dabei existiert eine feste Anzahl im PCW-System fest definierter Verarbeitungsmethoden. Der Anwendungscode der Consumer-App muss die Weiterverarbeitung nicht selbst implementieren. Windows stellt eine User-Mode-API für PCW-Consumer bereit, welche diese fest definierten Weiterverarbeitungsschritte im User-Mode durchführt. Die Weiterverarbeitung findet jedoch immer im Consumer nach dem Erhalt der Rohdaten aus dem Kernel statt. Ein Beispiel sind Frequenz-Werte (Raten). Die Rohdaten enthalten dabei die Anzahl bestimmter gemessener Ereignisse zwischen der Probe, welche die Daten enthält, und der vorangegangenen Probe. Der Consumer benutzt die Zeitstempel aus den zwei Proben, um den Zeitraum zwischen den zwei Proben zu ermitteln. Dann wird der Zählerwert durch den Zeitraum geteilt, um die Rate für eine feste Zeiteinheit zu ermitteln [Mic3a].

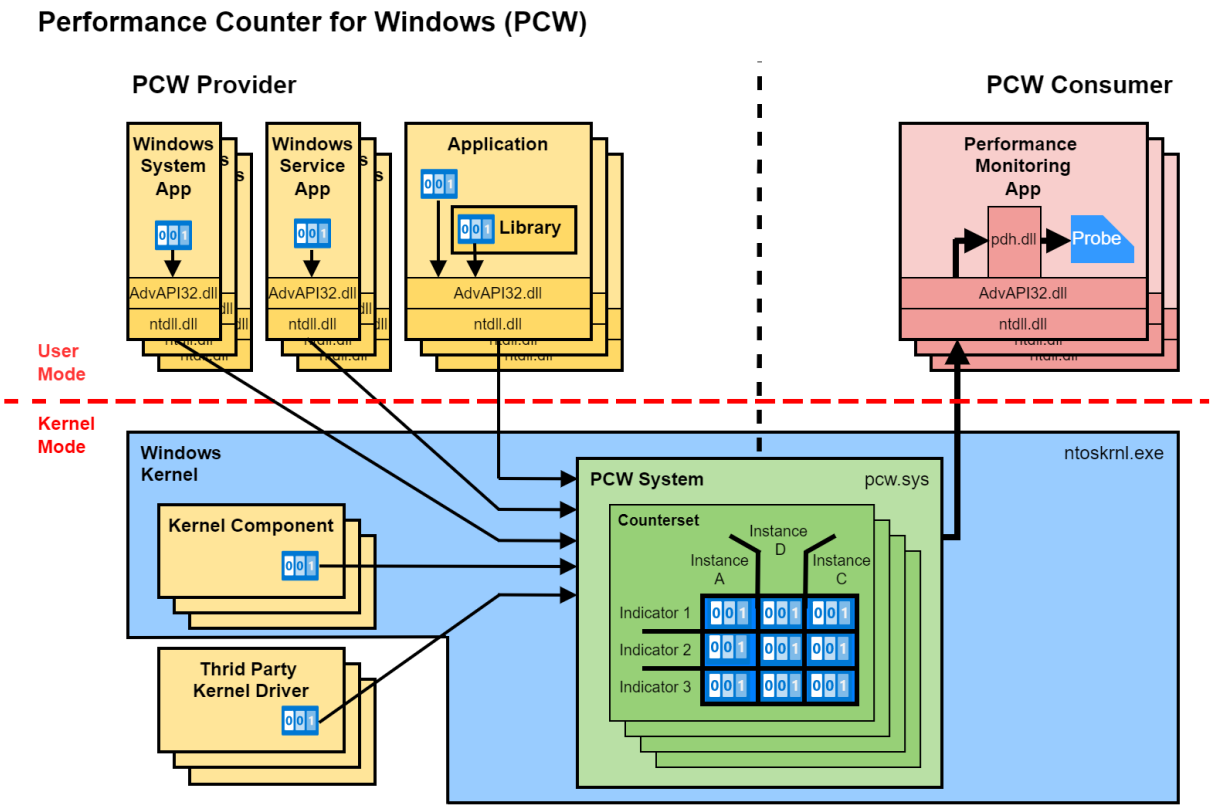

Abbildung 2.1: Performance Counter for Windows (PCW) - Architektur

2.2 Messung der Auftrittsfrequenzen von Ereignissen

Der Zweck dieser Messung ist es, eine grobe Einschätzung der Auftrittsfrequenzen bestimmter Basisereignisse zu liefern.

Dabei muss ausdrücklich gesagt werden, das die Auftrittsfrequenz für ein speziellen Ereignistyp variieren kann. Zum einen ist diese abhängig vom verwendeten Computersystem (Hardware, Software). Zum anderen sollten die Frequenzen deutlich in Abhängigkeit der Nutzung des Systems variieren. Darüber hinaus wird auch erwartet, dass die Auftrittsfrequenzen bei mehrfacher Messung bei möglichst gleichen Rahmenbedingungen dennoch leicht variieren. Letzteres liegt darin begründet, dass die Gleichheit der Rahmenbedingungen zwischen mehreren Messungen immer nur bis zu einem gewissen Grad hergestellt werden können. Tief im Detail ergeben sich immer Unterschiede im Ausführungsverhalten. Es wird jedoch angenommen, dass deren Einfluss gering ist.

Aus den genannten Gründen sind diese Messungen nicht dafür vorgesehen, exakte Werte zu liefern, da eine solche Exaktheit aus den oben genannten Gründen wenig allgemeine Aussagekraft besitzt. Stattdessen sollen die Messwerte, die für in etwa gleiche Rahmenbedingungen mehrmals in mehreren Messvorgängen hintereinander ermittelt werden, dazu dienen, die Größenordnung der Auftrittsfrequenzen von bestimmten Basisereignissen für bestimmte Nutzungssituationen abzuschätzen.

2.2.1 Messdurchführung

Die Auftrittsfrequenzen verschiedener Basisereignisse werden auf einem Windows-PC durch Performance-Counter-for-Windows (PCW) ermittelt. Beim Windows-PC handelt es sich um eine reale physische Maschine (keine VM) in Form eines Laptops. Die Hardware- und Software-Konfiguration ist in den Abschnitten „Hardware-System“ und „Software-System“ näher beschrieben.

Auf dem PC geschieht nach dem Booten des Windows-Betriebssystems eine Anmeldung mit dem Benutzerkonto des Standard-Benutzers. Danach wird die Windows-eigene Software `perfmon` gestartet. In dieser werden die PCW-Indikatoren für die zu ermittelnden Auftrittsfrequenzen eingestellt. Die PCW-Indikatoren sind im Abschnitt „Ereignistypen“ näher erläutert. Vor dem Start der Messung werden die Rahmenbedingungen für das Nutzungsszenario hergestellt (bestimmte Programme oder Vorgänge starten).

Die Messung wird aus der `perfmon`-Software heraus gestartet. Je nach Nutzungsszenario werden während der Messung entsprechende Eingaben getätigt. Die Nutzungsszenarien sind detailliert im Abschnitt „Nutzungsszenarien“ beschrieben. Während die Messung läuft, gehen im Sekundentakt die Werte der PCW-Indikatoren in der `perfmon`-Software

ein. Die Zeitdauer einer Messung ist eine für das entsprechende Nutzungsszenario eindeutig definierte Zeitdauer. Diese Zeitspannen liegen je nach Nutzungsszenario zwischen ca. 2 Minuten und 10 Sekunden. Die Präzision der Messdauer ist nicht besonders wichtig, da es sich bei allen genutzten PCW-Indikatoren bereits um Frequenzen (Raten) handelt.

Die `perfmon`-Software liefert für jeden PCW-Indikator unter anderem einen Graphen, indem der Verlauf der eingegangenen Werte über die Zeit visualisiert wird. Zudem werden die höchsten und niedrigsten gemessenen Werte innerhalb des Gesamtzeitraums der Messung angegeben. Es gibt zwei verschiedene Typen von Nutzungsszenarien, die unterschiedlich bemessen werden. Zum einen gibt es Nutzungsszenarien, die beliebig lange Vorgänge beschreiben (z.B. Idle, YouTube-Playback, Fensterbewegung). Bei diesen wird grob eine obere Schranke und eine untere Schranke visuell aus dem Graphen ermittelt. Dabei werden Peaks ausgeschlossen. Peaks sind einzelne Werte, die stark von den restlichen Werten abweichen. Zeitlich darf es sich dabei aber nur um einzelne Werte handeln ($<$ eine Sekunde). Peaks sollen deswegen herausgefiltert werden, da diese vermutlich eine andere Ursache haben und nicht charakteristisch für das Nutzungsszenario sind. Zum anderen gibt es Nutzungsszenarien die einen kurzen, zeitlich begrenzten Vorgang beschreiben, für den erwartet wird, dass bei diesem eine besonders hohe Ereignisfrequenz gemessen wird (z.B. Starten von Programmen). Bei diesen Messungen ist der von der `perfmon`-Software ermittelte Maximalwert das Ergebnis.

Für jedes Nutzungsszenario wird der Messvorgang drei mal durchgeführt, um herauszufinden, ob die Werte der drei Messungen im ähnlichen Bereich liegen. Es wird erwartet, dass dies so ist. Sollte dies so sein, wird das Endergebnis als eine Vereinigung der Bereiche der drei Messungen angegeben. Sollten die Wertebereiche zwischen den drei Messungen quasi gleicher Rahmenbedingungen stark voneinander abweichen, soll dies gesondert in den Ergebnissen angezeigt werden.

2.2.2 Hardware-System

Bei dem Computersystem, auf dem die Auftrittsfrequenzen ermittelt werden, handelt es sich um einen Laptop aus dem Jahre 2014.

Gerät:	Lenovo Thinkpad E540
Prozessor (CPU):	Intel Core i5-4210M
physische CPU-Kerne:	2
logische CPU-Kerne:	4 (SMT)
Basistakt:	2,6 GHz
Turbotakt:	3,2 GHz
LLC-Cache:	3 MB
Arbeitsspeicher:	8 GB
Datenträgertyp:	SSD

Tabelle 2.1: Ereignis-Frequenzmessungen: verwendete Hardware

2.2.3 Software-System

Auf dem Laptop befindet sich für die Messung als Betriebssystem ein frisch installiertes Windows 10 Pro 64 Bit 21H2. Das Betriebssystem ist zum Zeitpunkt der Messung vollständig aktualisiert (Alle Updates, die online bereitstehen, sind installiert). Dies soll auch einem unerwünschten Updatevorgang während einer Messung vorbeugen. Darüber hinaus sind die notwendigen Gerätetreiber installiert, sodass die internen Komponenten des Laptops korrekt arbeiten.

Zudem sind folgende Anwendungsprogramme auf dem Rechner installiert:

- LibreOffice (7.3.1)
- Mozilla FireFox Browser (91.6.1)
- Zoom (5.9.6.3799)

2.2.4 Ereignistypen

Im Folgenden sind die Ereignistypen aufgelistet, deren Auftrittsfrequenzen bemessen werden. Zu jedem Ereignistyp ist die PCW-Konfiguration bestehend aus Counterset, Indikator und den gewählten Instanzen (falls vorhanden) angegeben.

Ereignistyp	**PCW-Couterset**	**PCW-Indikator**	**Instanzen**
Interrupts (gesamt)	Prozessor	Interrupts/s	_Total
Interrupts (pro CPU-Kern)	Prozessor	Interrupts/s	0, 1, 2, 3
Timer-Interrupts (gesamt)	Prozessorinformationen	Zeitinterrupts/s	_Total
Timer-Interrupts (pro CPU-Kern)	Prozessorinformationen	Zeitinterrupts/s	0, 1, 2, 3
DPCs (gesamt)	Prozessor	DPC-Rate	_Total
DPCs (pro CPU-Kern)	Prozessor	DPC-Rate	0, 1, 2, 3
Threadwechsel	System	Kontextwechsel/s	-
Systemcalls	System	Systemaufrufe/s	-
Page-Faults (soft/hard)	Arbeitsspeicher	Seitenfehler/s	-
Hard Page-Faults	Arbeitsspeicher	Seiten/s	-
Objekt-Zugriffe	System	Dateivorgänge/s	-
Lesezugriffe auf Dateien	Logischer Datenträger	Lesevorgänge/s	_Total
Schreibzugriffe auf Dateien	Logischer Datenträger	Schreibvorgänge/s	_Total

Tabelle 2.2: Ereignis-Frequenzmessungen: verwendete Performance-Counter (PCW)

2.2.5 Nutzungsszenarien

Die Nutzungsszenarien sind in zwei Gruppen aufgeteilt. In der einen Gruppe befinden sich die Nutzungsszenarien, die Vorgänge beliebiger Länge beschreiben. In der anderen Gruppe sind kurzzeitige Vorgänge beschrieben.

2.2.5.1 Nutzungsszenarien beliebig langer Vorgänge

Szenario: Leerlauf (idle)

Vor der Messung werden alle Netzwerkverbindungen getrennt und der WLAN-Adapter in den Flugzeugmodus geschaltet. Während der Messung ist nur die `perfmon`-Anwendung gestartet, abgesehen aller Hintergrund-Prozesse, die standardmäßig aktiv sind. Es werden während der Messung keinerlei Eingaben durch den Benutzer getätigt. Die Messung dauert zwei Minuten.

Zweck dieses Szenarios ist die Grundaktivität also die Ereignisfrequenzen, die auftreten, wenn der PC nicht benutzt wird und nur die Standard-Hintergrundprozesse aktiv sind, zu ermitteln.

Szenario: Mausbewegung

Vor der Messung werden alle Netzwerkverbindungen getrennt und der WLAN-Adapter in den Flugzeugmodus geschaltet. Während der Messung ist nur die `perfmon`-Anwendung gestartet, abgesehen aller Hintergrund-Prozesse, die standardmäßig aktiv sind. Die Maus wird während der Messung kontinuierlich bewegt. Es wird nicht geklickt und es werden keine Tastatureingaben getätigt. Die Messung dauert zwei Minuten.

Der Zweck dieses Szenarios ist den Einfluss von Benutzereingaben über die Maus auf die Auftrittsfrequenzen zu erkennen. Bewegt sich der Zeiger über die Benutzeroberfläche werden Prozesse benachrichtigt, welche die Maus-Ereignisse behandeln. Zudem kommt es unter Umständen zum Aktualisieren der Oberfläche. Es wird daher erwartet, dass eine größere Menge an Systemcalls für die Kommunikation der Anwendungs-Prozesse und des Fenstermanager-Prozesses (DWM) mit der im Kernel befindlichen GDI-Schnittstelle auftreten. Zudem sollte die Aktivität des I/O-Gerätes Maus in der Menge an auftretenden Hardware-Interrupts und erzeugten DPCs sichtbar werden. Die Größenänderung bezüglich der verschiedenen Ereignisfrequenzen im Vergleich zum Idle-Zustand abzuschätzen soll zusammengefasst Zweck dieser Messungen sein.

Szenario: Fensterbewegung

Vor der Messung werden alle Netzwerkverbindungen getrennt und der WLAN-Adapter in den Flugzeugmodus geschaltet. Während der Messung ist nur die `perfmon`-Anwendung gestartet, abgesehen aller Hintergrund-Prozesse, die standardmäßig aktiv sind. Das nicht maximierte Fenster der `Perfmon`-Software wird während der Messung kontinuierlich verschoben. Der Fensterrenderer in Windows ist standardmäßig so konfiguriert, dass der Fensterinhalt während der Bewegung des Fensters permanent neu gezeichnet wird und somit auch während der Bewegung sichtbar ist. Die Messung dauert zwei Minuten.

Der Zweck dieses Nutzungsszenarios ist ähnlich dem vorangegangenen. Durch das Verschieben eines Fensters ist es in der Regel notwendig, dass der Inhalt durch die Software neu gezeichnet wird. Es wird vermutet, dass dies eine größere Last verursachen und sich durch höhere Auftrittsfrequenzen bestimmter Ereignistypen bemerkbar macht.

Szenario: Video-Stream im Browser

Vor der Messung wird die Verbindung zum Internet hergestellt. Dies geschieht über eine WLAN-Verbindung zu einem lokalen Funk-Netzwerk, das mit dem Internet verbunden ist. Zudem wird der installierte Browser Mozilla FireFox geöffnet. Es wird sichergestellt, dass nur ein Browser-Tab geöffnet ist. Mit diesem Browser-Tab wird das Web-Portal YouTube angesteuert und ein Video mit ausreichender Länge (2 Minuten Messung) gestartet. Während der Messung ist neben den Hintergrund-Prozessen nur die `perfmon`-Anwendungen und der FireFox-Browser gestartet. Beide Fenster werden nebeneinander,

nicht maximiert auf dem Bildschirm angezeigt. Während der gesamten Messung befindet sich das Video dauerhaft im Abspiel-Modus. Das Video ist dabei nicht maximiert, sondern wird eingebettet im restlichen Webseiteninhalt im Browserfenster dargestellt. Die Messung dauert zwei Minuten.

Dieses Szenario beschreibt einen echten Anwendungsfall der PC-Nutzung durch Endanwender. Beim Video-Stream im Browser ist der Netzwerk-Stack aktiv, um die Streaming-Daten zu empfangen und den Empfang zu bestätigen. Das komprimierte Video muss mittels einer Codec-Software (geladene Bibliothek im Browser-Prozess) dekomprimiert werden und dann in der Oberfläche dargestellt werden. Letzteres geschieht innerhalb des FireFox-Prozesses und dessen Kindprozessen. Zwischen den Eltern- und Kindprozessen findet unter anderem Kommunikation über die Windows-IPC-Technik DCOM (nutzt RPC) statt. Es wird vermutet, dass die beschriebenen Vorgänge in den Auftrittsfrequenzen deutlich sichtbar werden. Es soll mit diesem Anwendungsfall die Größenordnung der Auftrittsfrequenz für diesen realen Anwendungsfall abgeschätzt werden, der stellvertretend für ähnliche Anwendungsfälle, bei denen Audio/Video-Playback stattfindet, steht.

Szenario: Video-Telefonie über Zoom

Vor der Messung wird die Verbindung zum Internet hergestellt. Dies geschieht über eine WLAN-Verbindung zu einem lokalen Funk-Netzwerk, das mit dem Internet verbunden ist. Die vorher installierte Meeting-Software Zoom wird geöffnet. Es wird ein Meeting mit einem weiteren Rechner instanziiert. Beide Maschinen senden ihren Video- und Audiostream (Webcam, Mikrofon). Die Ansicht ist auf den Galerie-Modus eingestellt, sodass beide Video-Streams dargestellt werden. Während der Messung ist neben den Hintergrund-Prozessen nur die `perfmon`-Anwendungen und die Zoom-Software gestartet. Beide Fenster werden nebeneinander nicht maximiert auf dem Bildschirm angezeigt. Die Messung dauert zwei Minuten.

Dieses Szenario beschreibt wiederum, einen echten Anwendungsfall der PC-Nutzung durch einen Endanwender. Bei der Video-Telefonie sind ähnliche Vorgänge, wie beim zuvor beschrieben Nutzungsszenario involviert. Jedoch werden mit Mikrofon und Webcam zwei weitere Eingabegeräte genutzt. Es wird erwartet, dass dies die Ereignisfrequenzen von Ereignissen, die im Zusammenhang mit I/O stehen (Interrupts, DPCs, Kontextwechsel, Systemcalls) erhöhen sollte. Zudem müssen die versendeten Daten der Streams vorher enkodiert werden, was sich in einer höheren Auslastung des Systems bemerkbar machen sollte. Alles in allem sind auch für diesen realen Anwendungsfall die Auftrittsfrequenzen der Ereignisse sehr interessant.

Szenario: Installation einer Software

Während dieser Messung ist die Verbindung zum Internet hergestellt. Das zuvor heruntergeladene Installationspaket für die Software LibreOffice wird gestartet und der Dialog wird bis zum Installationsvorgang durchgeführt. Dieser wird vor dem Start der Messung gestartet. Während der Installationsvorgang läuft wird die zweiminütige Messung gestartet. Der Installationsvorgang besitzt auf dem Testsystem eine längere Laufzeit als die zweiminütige Messung. Während der Messung ist neben den Hintergrund-Prozessen nur die Installationsanwendung und die `perfmon`-Applikation gestartet. Die Messung dauert zwei Minuten.

Die Installation einer Software ist ein Anwendungsfall, von dem klassischerweise eine eher höhere Systemauslastung erwartet wird. Während des Installationsprozesses werden komprimierte Daten aus dem Installationspaket gelesen, dekomprimiert und in Form vieler verschiedener Dateien auf den Datenträger geschrieben. Zudem wird die Systemkonfiguration angepasst und erweitert, indem eine größere Menge an Registry-Einträgen gelesen, bearbeitet und erstellt werden. Die Abarbeitung dieser Vorgänge wird in der Regel nicht durch das Warten auf Timer-Ereignisse oder Benutzereingaben gebremst, wodurch es hauptsächlich nur zu Wartevorgängen hinsichtlich der Datenträger-Interaktion kommt. Daher wird für den Installationsvorgang eine höhere Auftrittsfrequenz einiger Ereignistypen erwartet. Im Unterschied zu den vorangegangenen Szenarien wird bei diesen Szenario auch eine erhöhte Ereignisrate bezüglich Datei- und Objekt-Zugriffen erwartet.

2.2.5.2 Nutzungsszenarien mit Einzelaktionen

Bei den Nutzungsszenarien die nur aus einer zeitlich begrenzten Einzelaktion bestehen, handelt es sich ausschließlich um die Startvorgänge verschiedener Anwendungen. Der Startvorgang von Applikationen bzw. die Prozesserstellung ist deshalb so interessant, da diese erwartungsgemäß einen Peak in Bezug auf die Menge der gemessenen Systemvorgänge pro Sekunde hervorrufen wird. Beim Starten eines Prozesses müssen das Executable und etliche weitere Bibliotheksmodule geladen werden. Dies verursacht Dateizugriffe, sollten die Module nicht bereits geladen sein. Zudem werden bei der Initialisierung eine größere Menge an Systemcalls getätigt. Diese gehen von der Laufzeitumgebung wie auch vom Anwendungscode selbst aus. Zum Beispiel lesen Programme beim Start große Mengen ihrer Konfiguration ein. Dies beinhaltet Zugriffe auf Konfigurationsdateien und Registry-Schlüssel. Läuft der Prozess an, kommt es zudem beim Zugriff auf den neu erzeugten virtuellen Adressraum aufgrund des Demand-Paging zu einer Häufung von Page-Faults, deren Behandlung wiederum Systemaktionen und Dateizugriffe verursacht. Daher wird für den Startvorgang von Anwendungen ein besonders hoher Peak bezüglich

der Auftrittsfrequenzen erwartet. Diesen in seiner Größe abzuschätzen soll zusammenfassend Zweck dieser Messungen sein.

Vor der Messung ist sicherzustellen, dass die Netzwerkverbindung hergestellt ist und nur die `perfmon`-Anwendung gestartet ist. Somit wird vor dem Beginn der Messung neben dem `perfmon`-Prozess nur die Standard-Hintergrundprozesse ausgeführt. Es wird pro Messung eine Software gestartet. Die Messung wird dabei kurz vor dem Startvorgang der Anwendung über die `perfmon`-Oberfläche gestartet. Direkt nach dem Start der Messung wird die entsprechende Anwendung gestartet. Die Dauer der Messung soll 10 Sekunden betragen. Die Dauer der Startvorgänge der gewählten Anwendungen auf dem Testsystem ist kürzer als die Messdauer von 10 Sekunden.

Der Startvorgang folgender Anwendungen soll bemessen werden:

- Windows Explorer
- LibreOffice Writer
- Mozilla FireFox
- Zoom

2.3 Messergebnisse

Leerlauf (idle)	**Mausbewegung**	**Fensterverschiebung**
< 1% Auslastung @ 800 MHz	2% - 3% Auslastung @ 800 MHz	7% - 10% Auslastung @ 800 MHz
Interrupts (gesamt)		
280/s - 400/s	1.300/s - 1.600/s	2.000/s - 2.600/s
3.571µs - 2.500µs	769µs - 625µs	500µs - 385µs
Interrupts (pro Kern)		
20/s - 200/s	20/s - 1.000/s	100/s - 1.300/s
50.000µs - 5.000µs	50.000µs - 1.000µs	10.000µs - 769µs
Timer-Interrupts (gesamt)		
60/s	75/s	100/s
17ms	13ms	10ms
Timer-Interrupts (pro Kern)		
15/s	19/s	25/s
67ms	53ms	40ms
DPCs (gesamt)		
80/s - 200/s	250/s - 320/s	300/s - 400/s
13ms - 5ms	4ms - 3ms	3ms - 2ms
DPCs (pro Kern)		
0/s - 100/s	0/s - 300/s	0/s - 400/s
> 10ms	> 3ms	> 2ms
Systemcalls		
400/s - 1.800/s	15.000/s - 18.000/s	30.000/s - 50.000/s
2.500µs - 556µs	67µs - 56µs	33µs - 20µs
Kontextwechsel		
250/s - 400/s	1.100/s - 1.300/s	1.200/s - 1.400/s
4.000µs - 2.500µs	909µs - 769µs	833µs - 714µs
All Page Faults		
0/s - 100/s	0/s - 100/s	0/s - 100/s
> 10ms	> 10ms	> 10ms
Hard Page Faults		
0/s - 1/s	0/s - 1/s	0/s - 1/s
> 1s	> 1s	> 1s
Objekt-Zugriffe allgemein (Dateien, Registry, IPC, Sockets usw.)		
0/s - 30/s	120/s - 150/s	120/s - 150/s
> 33ms	8ms - 6ms	8ms - 6ms
Dateivorgänge auf Datenträgern (lesen)		
0/s - 2/s	0/s - 2/s	0/s - 2/s
> 500ms	> 500ms	> 500ms
Dateivorgänge auf Datenträgern (schreibend)		
0/s - 6/s	0/s - 6/s	0/s - 6/s
> 167ms	> 167ms	167ms

Tabelle 2.3: Ereignis-Frequenzmessungen: Messergebnisse - Vorgänge 1

Messergebnisse Langzeit-Vorgänge (Teil 2)

Leerlauf (idle)	YouTube im FireFox	Zoom-Video-Meeting	LibreOffice-Installation
< 1% @ 800 MHz	10% - 15% @ 1 GHz	12% - 18% @ 1,24 GHz	34% - 40% @ 3,2 GHz
Interrupts (gesamt)			
280/s - 400/s	6.000/s - 7.000/s	6.500/s - 7.500/s	10.000/s - 30.000/s
3.571µs - 2.500µs	167µs - 143µs	154µs - 133µs	100µs - 33µs
Interrupts (pro Kern)			
20/s - 200/s	1.000/s - 2.000/s	1.500/s - 2.500/s	1.000/s - 4.000/s
50.000µs - 5.000µs	1.000µs - 500µs	667µs - 400µs	1.000µs - 250µs
Timer-Interrupts (gesamt)			
60/s	400/s - 600/s	1.600/s - 1.900/s	400/s
16.667µs	2.500µs - 1.667µs	625µs - 526µs	2.500µs
Timer-Interrupts (pro Kern)			
15/s	100/s - 150/s	400/s - 475/s	100/s
66.667µs	10.000µs - 6.000µs	2.500µs - 2.105µs	10.000µs
DPCs (gesamt)			
80/s - 200/s	500/s - 1.100/s	750/s - 850/s	1.000/s - 2.800/s
12.500µs - 5.000µs	2.000µs - 909µs	1.333µs - 1.176µs	1.000µs - 357µs
DPCs (pro Kern)			
0/s - 100/s	20/s - 600/s	40/s - 700/s	500/s - 1.300/s
> 10.000µs	50.000µs - 2.000µs	25.000µs - 1.000µs	2.000µs - 769µs
Systemcalls			
400/s - 1.800/s	26.000/s - 30.000/s	15.000/s - 20.000/s	100.000/s - 120.000/s
2.500µs - 556µs	38µs - 33µs	67µs - 50µs	10µs - 8µs
Kontextwechsel			
250/s - 400/s	10.000/s - 12.000/s	8.200/s - 8.900/s	10.000/s - 60.000/s
4.000µs - 2.500µs	100µs - 83µs	122µs - 112µs	100µs - 17µs
All Page Faults			
0/s - 100/s	1.500/s - 4.500/s	0/s - 100/s	4.000/s - 15.000/s
> 10.000µs	667µs - 222µs	> 10.000µs	250µs - 67µs
Hard Page Faults			
0/s - 1/s	0/s - 30/s	0/s - 1/s	10/s - 15/s
> 1.000ms	> 33ms	> 1.000ms	100ms - 67ms
Objekt-Zugriffe allgemein (Dateien, Registry, IPC, Sockets usw.)			
0/s - 30/s	2.900/s - 3.100/s	15/s - 40/s	10.000/s - 20.000/s
> 33.000µs	345µs - 323µs	66.667µs - 25.000µs	100µs - 50µs
Lesevorgänge auf Dateien (Datenträger)			
0/s - 2/s	0/s - 140/s	0/s - 2/s	80/s - 120/s
> 500ms	> 7ms	> 500ms	13ms - 8ms
Schreibvorgänge auf Dateien (Datenträger)			
0/s - 6/s	0/s - 30/s	30/s - 35/s	300/s - 600/s
> 167ms	> 33ms	33ms - 29ms	3ms - 2ms

Tabelle 2.4: Ereignis-Frequenzmessungen: Messergebnisse - Vorgänge 2

Messergebnisse Startvorgang Anwendungen (Peak-Werte)

Windows Explorer	LibreOffice Writer	Firefox	Zoom
43% @ 3 GHz	70% @ 3,1 GHz	100% @ 3 GHz	40% @ 2,8 GHz
Interrupts (gesamt)			
12.001/s	25.049/s	12.513/s	11.877/s
83µs	39µs	79µs	84µs
Interrupts (pro Kern)			
1.000/s - 5.000/s	3.500/s - 10.000/s	3.000/s - 5.000/s	2.000/s - 4.000/s
1.000µs - 200µs	285µs - 100µs	333µs - 200µs	500µs - 250µs
Timer-Interrupts (gesamt)			
172/s	373/s	1.400/s	1.760/s
5.814µs	2.680µs	714µs	568µs
Timer-Interrupts (pro Kern)			
43/s	93/s	350/s	440/s
23.256µs	10.753µs	2.857µs	2.273µs
DPCs (gesamt)			
777/s	1.176/s	1.006/s	1.234/s
1.287µs	850µs	994µs	810µs
DPCs (pro Kern)			
100/s - 500/s	70/s - 800/s	100/s - 900/s	70/s - 300/s
10ms - 2ms	14ms - 1ms	10ms - 1ms	14ms - 3ms
Systemcalls			
234.610/s	122.660/s	149.192/s	102.516/s
4µs	8µs	7µs	10µs
Kontextwechsel			
24.884/s	12.282/s	28.982/s	15.040/s
40µs	81µs	35µs	66µs
All Page Faults			
10.058/s	136.493/s	118.869/s	33.887/s
100µs	7µs	8µs	30µs
Hard Page Faults			
1.324/s	4.636/s	29.133/s	9.058/s
755µs	216µs	34µs	110µs
Objekt-Zugriffe allgemein (Dateien, Registry, IPC, Sockets usw.)			
1.326/s	5.535/s	8.495/s	643/s
754µs	180µs	117µs	1.555µs
Lesevorgänge auf Datei (Datenträger)			
86/s	796/s	981/s	405/s
11ms	1ms	1ms	2ms
Schreibvorgänge auf Dateien (Datenträger)			
317/s	123/s	602/s	150/s
3ms	8ms	2ms	7ms

Tabelle 2.5: Ereignis-Frequenzmessungen: Messergebnisse - Software-Starts (Peak-Werte)

2.4 Interpretation der Messergebnisse

Vorab muss zur korrekten Interpretation der Messwerte die Information gegeben werden, dass zu jeder Frequenzangabe auch die Periodendauer angegeben ist. Diese ist jedoch nur dafür gedacht, die Frequenz besser einordnen zu können. Die genannte Periodendauer ist direkt als reziproker Wert aus der Auftrittsfrequenz errechnet. Sie gibt keine Aussage darüber, in welchen Abständen zwei aufeinanderfolgende Ereignisse gleichen Typs tatsächlich stattfinden (absehen vom Timer-Ereignis). Z.B. können zwei Systemcalls auch in einem viel kürzeren zeitlichen Abstand hintereinander auftreten als die genannte Periodendauer. Somit ergibt sich durch die Auftrittsfrequenzen eher eine Abschätzung bezüglich des Workloads und der Datenmenge, wollte man eingehende Event-Objekte diesen Typs verarbeiten. Eine Aussage über das vorhandene Zeitfenster, bis das nächste Ereignis eintritt, kann aus den Messwerten nicht ermittelt werden.

Das Szenario Leerlauf liefert Messwerte, die das „Grundrauschen“ eines Windows-PCs abbilden. Aus den Auftrittsfrequenzen für die verschiedenen Systemereignisse kann abgeschätzt werden, wie viele Ereignisse bereits im Leerlauf auf einem Windows-PC auftreten. Die Basisgeräte eines Laptops ohne aktive Netzwerkverbindung verursachen in den durchgeführten Messungen bereits 280 bis 400 Interrupt-Anfragen pro Sekunde. 60 Interrupts pro Sekunden entfallen alleine auf die System-Timer-Hardware. Die Interrupt-Service-Routinen der Gerätetreiber erzeugen 80 bis 200 DPCs pro Sekunde, um die IO-Geräte zu steuern. Auf einem frisch installierten Windows-PC werden, ohne das explizit durch den Benutzer eine Anwendung gestartet wurde, bereits ca. 170 Prozesse und ca. 2.100 Threads ausgeführt. Diese verursachen im Leerlauf 400 bis 1.800 Systemcalls pro Sekunde. Zwischen ihnen wird im Leerlauf 250 bis 400 mal in der Sekunde gewechselt (Kontextwechsel). Im Leerlauf geschehen nur sehr wenige Page-Faults, da diese hauptsächlich beim Starten von Anwendungen oder bei der Durchführung Speicher-intensiver Vorgänge auftreten. Die im Leerlauf gemessene Anzahl an Page-Faults liegt zwischen 0 und 100 pro Sekunde. In der Regel handelt es sich dabei aber nicht um Hard-Page-Faults, die einen Zugriff auf den Datenträger mit sich bringen. Die Anzahl an auftretenden Hard-Page-Faults liegt im Leerlauf bei weniger als einem pro Sekunde. Prozesse greifen im Leerlauf zum Ausführen ihrer Hintergrund-Aktivitäten bis zu 30 mal in der Sekunde auf System-Objekte zu. Dazu zählen jegliche Zugriffsvorgänge wie z.B. Zugriffe auf Dateien, die Registry oder IPC-Konstrukte. Die Zugriffe auf persistent gespeicherte Dateien auf Datenträgern sind noch geringer. Hier wurden im Leerlauf maximal 2 lesende Zugriffe pro Sekunde und 6 schreibende Zugriffe pro Sekunde beobachtet.

Im ersten Teil der Vorgangsmessungen wurden die Auftrittsfrequenzen für die drei Szenarien Leerlauf, Mausbewegung und Fensterbewegung ermittelt. Da bei den Szenarien Mausbewegung und Fensterbewegung abgesehen von IO-Vorgängen (Mauseingaben,

Bildschirmausgabe) nur Aktionen im Bereich der Grafischen-Benutzeroberfläche geschehen, wird hier ein interessanter Aspekt bezüglich der Systemcall-Menge sichtbar. Kommunikation zwischen Anwendungsprozessen und dem Kernel über Systemcalls, die sich auf die Benutzerinteraktion via GUI (Fenster) beziehen, werden von `User32.dll` und GDI32.dll durchgeführt und im Kernel (`ntoskrl.exe`) vom System-Service-Dispatcher an den Win32-Kernel (`win32k.sys`) weitergeleitet, der sich gemeinsam mit dem User-Mode-Prozess Desktop-Window-Manager (`dwm.exe`) um das Fenstermanagement und die GUI kümmert. Somit kann man zwischen zwei Kategorien von Systemcalls unterscheiden. Diese, die sich auf die grafische Benutzeroberfläche (GUI) und das Fenstermanagement beziehen und die klassischen Systemcalls, die vom Code der `NTDLL.dll` durchgeführt werden, sich auf Zugriffe auf System-Ressourcen beziehen und vom Windows-Kernel (`ntoskrnl.exe`) verarbeitet werden. Ein Vergleich der Auftrittsfrequenzen für Systemcall-Ereignisse zwischen den Szenarien Leerlauf, Mausbewegung und Fensterbewegung ermöglichen das Verhältnis zwischen den zwei genannten Systemcall-Klassen abzuschätzen. Im Leerlauf liegt die Anzahl an Systemcalls zwischen 400/s und 1.800/s, während sie alleine durch die Mausbewegung auf 15.000/s bis 18.000/s ansteigt. Somit ist zu Vermuten, dass alleine die Vorgänge des Neuzeichnens des Mauszeigers auf der Oberfläche und die Benachrichtigung der Prozesse, die Fenster registriert haben, über Aktualisierungen der Zeiger-Position, ca. 15.000 Systemcalls pro Sekunde aus. Durch die Benachrichtigungen der Prozesse mit registrierten Fenstern durch den Win32-Kernel (`win32k.sys`), werden die GUI-Threads der Prozesse geweckt und werden aktiv, bis sie wieder blockieren. Dies erhöht insgesamt die Menge an Kontextwechseln. Genau dies kann in den Messergebnissen beobachtet werden. Die Anzahl der Kontextwechsel steigt von 250/s bis 400/s im Leerlauf auf 1.100/s bis 1.300/s bei Mausbewegung. Wird das Fenster gehalten und verschoben, erhöht sich die Anzahl von Benachrichtigungen und Aufforderungen seitens des Prozesses, der das Fenster registriert hat, das verschoben wird, sowie der Prozesse, deren Fenster durch das verschobene Fenster verdeckt oder aufgedeckt werden, ihre Fensterinhalte neu zu zeichnen. So steigt die Anzahl der Systemcalls in diesem Fall auf 30.000/s bis 50.000/s an. Die Systemcalls, die auf das Win32-Fenstersystem entfallen (`Win32k.sys`) sind somit vermutlich ca. 40.000/s. Der Anteil klassischer Systemcalls, die sich nicht auf die Benutzeroberfläche beziehen, würde somit nur einen Anteil von 7% ausmachen. Diese Information ist sehr relevant für die Abschätzung bezüglich des Overheads einer Überwachungssensorik, die auf Systemcall-Ereignissen basiert. Da Vorgänge bezüglich der Benutzeroberfläche in der Regel für eine Endpunkt-Sensorik oder Angriffserkennung nicht relevant sind, müsste nur ein kleiner Teil der Gesamtmenge, der unter Windows auftretenden Systemcalls, analysiert werden. Diese Vermutung wird bekräftigt durch die Messwerte bezüglich der Zugriffe auf System-Objekte und Dateien. Diese bleiben weitestgehend über die drei verschiedenen Szenarien gleich. Nur die Anzahl der Objekt-Zugriffe steigt leicht im Vergleich zum Leerlauf an.

Da bei den Vorgängen Leerlauf, Mausbewegung und Fensterbewegung keine besonderen IO-Zugriffe explizit durch den Benutzer befohlen werden und die Verbindung zum Netzwerk getrennt ist (LAN-Kabel nicht angesteckt und Flugzeugmodus) kann durch diese Vorgänge auch der Einfluss durch IO-Geräte abgeschätzt werden. Im Leerlauf ist das „Hintergrund-Rauschen“ zu sehen, dass durch sämtliche Basisgeräte in einem PC-System verursacht wird. Die IO-Geräte verursachen im Leerlauf 280 bis 400 Interrupt-Anfragen pro Sekunde. Davon entfallen alleine 60 Interrupts pro Sekunde auf den System-Timer. Die Verteilung der Interrupts auf die CPU ist sehr unterschiedlich und schwankt zwischen 20/s und 200/s. Windows versucht die Zuteilung der Interrupt-Quellen auf Rechenkerne über den Interrupt-Controllers (APIC) dynamisch so zu konfigurieren, dass schlafende Rechenkerne nicht aufgeweckt werden, wenn dies nicht sein muss, und noch andere nicht zu stark ausgelastete CPU-Kerne zur Verfügung stehen [AIRS21, S. 66ff]. Dies verringert den Energieverbrauch und die Wärmeentwicklung. Die Interrupt-Service-Routinen (ISRs) der Gerätetreiber verursachen im Leerlauf 80 bis 200 DPCs pro Sekunde. Wird das Leerlauf-Szenario mit dem Mausbewegungsszenario oder dem Fensterverschiebungsszenario verglichen, ist ein Anstieg in der Interrupt-Frequenz zu erkennen. Da die Rahmenbedingungen gleich geblieben sind, kann davon ausgegangen werden, dass die zusätzlichen Interrupts auf die IO-Geräte Maus, USB-Controller, Grafikkarte und Bildschirmschnittstelle zurückzuführen sind. Der IO-Overhead für die Benutzerinteraktion mit der grafischen Benutzeroberfläche lässt sich damit abschätzen. Wird die Maus bewegt und der Zeiger auf dem Bildschirm aktualisiert, führt dies zu einem Anstieg um ca. 1.000 Interrupt-Anfragen pro Sekunde, die vermutlich auf die genannten IO-Geräte entfallen. Je größer die Aktivität ist, desto größer ist der Anstieg. Bei der Fensterbewegung konnte ein Anstieg um ca. 2.000 Interrupts pro Sekunde im Vergleich zum Leerlauf beobachtet werden. IO-Geräten verursachen im Leerlauf somit in etwa nur 5% der Interrupt-Menge die auftritt, wenn Benutzerinteraktion via Mauseingaben und Bildschirmausgaben getätigt werden. Die Anzahl der von ISRs der Gerätetreiber erzeugten DPCs stieg signifikant um ca. 200/s an im Vergleich zum Leerlauf-Szenario.

Die Messergebnisse bezüglich der Auftrittsfrequenzen des Timer-Interrupts bestätigen Messtechnisch den theoretisch beschriebenen technischen Sachverhalt der dynamischen Anpassung der Timer-Frequenz. Das Windows-Betriebssystem ändert die Konfiguration der Timer-Hardware im laufenden Betrieb und variiert dadurch dynamisch die Timer-Frequenz. Über die Messungen ist zu beobachten, dass das Timer-Interrupt zwischen 60 mal pro Sekunde und 2000 mal pro Sekunde variiert, sich jedoch nicht außerhalb dieser Grenzen bewegt. Zudem veränderten sich die beobachteten Werte nicht kontinuierlich sondern Stufenweise. Dies bestätigt die technische Dokumentation, die beschreibt, das Windows die Konfiguration des System-Timers im laufenden Betrieb variiert. Je nach Auslastung und Anforderung bezüglich der zeitlichen Auflösung, verändert Windows

die Frequenz der Timer-Hardware stufenweise zwischen 60 Hz und 2 KHz [AIRS21, S. 66ff].

Im zweiten Teil der Vorgangsmessungen wurden die Auftrittsfrequenzen für die Ausführung bestimmter Anwendungen gemessen, um somit die Ereignisraten für reale Workloads abzuschätzen. Das Streamen eines YouTube-Videos im FireFox-Browser verursacht dabei scheinbar viele Systemcalls. Die Anzahl stieg von 400/s bis 1.800/s im Leerlauf auf 26.000/s bis 30.000/s. Ein Zoom-Meeting verursachte nicht mehr Systemcalls sondern mit 15.000/s bis 20.000/s etwas weniger. Die Menge der Systemcalls entfällt sehr wahrscheinlich auf die Darstellung der Video-Stream-Inhalte auf der Oberfläche und der Kommunikation und Synchronisation zwischen den Threads und Kindprozessen der Anwendungen. Insbesondere Browser wie z.B. FireFox und Chrome verwenden hier aus Sicherheitsgründen ein Konstrukt aus mehreren isolierten Kindprozessen, die miteinander über IPC-Kanäle kommunizieren. Dies verursacht zum einen Systemcalls und zum anderen durch das Blockieren und Reaktivieren der Threads Kontextwechsel. Die Rate an Kontextwechseln steigt während des Video-Streamings/-Telefonie ebenfalls um etwa das 50-fache im Vergleich zum Leerlauf auf 10.000/s bis 12.000/s. Beim Installationsvorgang der LibreOffice-Suite wurde auf dem System hingegen die größte Rate von 100.000 bis 120.000 Systemcalls pro Sekunde gemessen. Dies verwundert nicht, da bei einer Installation eine große Menge von Dateien und Registry-Einträgen neu angelegt und geschrieben werden. Zudem müssen die zu installierenden Daten aus Archiven gelesen werden und System-Konfigurationsparameter abgefragt werden. Dies beinhaltet eine große Menge an Zugriffen auf System-Objekte wie Dateien und Registry-Schlüssel. Dies kann auch beobachtet werden, so ist die auf dem System gemessene Rate an Zugriffs-Operationen auf System-Objekte mit 10.000/s bis 20.000/s 1000 mal größer als im Leerlauf. Von diesen Zugriffen bezieht sich jedoch nur eine Minderheit auf reale Dateien auf Datenträgern. So wurde während des Installationsprozessen 80 bis 120 Lesevorgänge pro Sekunde und 300 bis 600 Schreibvorgänge pro Sekunde auf Dateien gemessen. Dies ist aber dennoch etwa 100 mal mehr als im Leerlauf. Eine aktive Zoom-Video-Konferenz verursacht hingegen nur sehr wenige Objekt-Zugriffe. Die Messwerte für das Gesamtsystem lagen bei 15/s bis 40/s. Das ist nur unwesentlich mehr als im Leerlauf. Von diesen Zugriffen entfällt dazu die größte Anzahl auf Datei-Schreibvorgänge auf den Datenträger (30/s bis 35/s). Datei-Lesevorgänge wurden hingegen kaum beobachtet. Es scheint so, dass der Zoom-Prozess, während das Meeting läuft, Protokollierungen vornimmt. Beim YouTube-Video-Stream im FireFox-Browser wurden hingegen auf dem System 2.900 bis 3.100 Objekt-Zugriffe pro Sekunde gemessen. Davon entfallen bis zu 140/s auf Datei-Lesevorgänge und bis zu 30/s auf Datei-Schreibvorgänge. Erklärt werden könnten diese durch Zugriffe auf die persistierten Web-Cache- und Cockie-Datenbanken durch den Firefox-Browser. Die restlichen Objekt-Zugriffe, die um den Faktor 10 größer sind, lassen sich vermutlich auf die Zugriffe auf die erwähnten IPC-Kanäle zurückführen. Die Rate der Page-Faults fällt

zwischen dem Video-Meeting via Zoom und dem Video-Streaming im FireFox-Browser verschieden aus. Während beim Zoom-Call keine größere Menge an Page-Faults auftritt (0/s bis 100/s) als im Leerlauf, verursacht das System mit laufenden FireFox-Prozessen 1.500 bis 4.500 Page-Faults pro Sekunden. Eventuell entstehen diese durch einen Speicherseitenbedarf, der durch die Video-Kompressionsvorgänge verursacht wird. Nur bei einer Minderheit dieser handelt es sich jedoch um Hard-Page-Faults (bis zu 30/s). Wie zu erwarten erzeugt die Dekompression sowie der Transfer großer Datenmengen während des Installationsprozesses ebenfalls einen Bedarf an Speicherseiten, der in Page-Fault-Ereignissen resultiert. Während des Installationsprozesses verursacht das System 4.000 bis 15.000 Page-Faults pro Sekunde. Dies sind in etwa 100 mal so viele im Vergleich zum Leerlauf.

Werden die Messergebnisse aller Szenarien verglichen, bestätigt sich die Erwartung, dass sehr hohe Auftrittsfrequenzen meist kein Dauerzustand sind, sondern als Peak auftreten, wenn besondere kurzzeitige Vorgänge durchgeführt werden, die das System besonders auslasten. Angelehnt an dieser Erwartung wurden die Startvorgänge von Anwendungsprogrammen gemessen, da diese generell als besonders arbeitsintensiv für ein Computersystem angesehen werden. Die Messungen bestätigen diese Vermutung. Während des kurzzeitigen Startvorgangs wurden extrem viel höhere Auftrittsfrequenzen der verschiedenen Ereignisse gemessen, als bei den Langzeit-Vorgängen (Dauerzuständen). Zusätzlich war zu beobachten, dass während der gemessenen Startvorgängen die CPU-Auslastung stark anstieg und der Prozessor auf seine maximale Taktfrequenz (Turbo-Takt) eingestellt wurde. Die Rate an Systemcalls stieg bei Startvorgängen von Anwendungen kurzzeitig auf Werte zwischen 100.000/s und 235.000/s. Dies ist ein Anstieg um mehr als das Tausendfache im Vergleich zum Leerlauf. Auch die Anzahl der Kontextwechsel stieg zwar ebenfalls, letztendlich fiel diese aber nicht viel größer aus, als bei den Szenarien YouTube-Stream und Zoom-Call. Sie war sogar kleiner als die Rate, die während des Installationsprozesses gemessen wurde. Während des Startvorgangs einer Software geschieht neben der Instanziierung durch das Betriebssystem und dem Laden der dynamischen Bibliotheken, auch die Initialisierung durch den Anwendungscode, bei der unter anderem die Konfiguration und Start-Ressourcen der Anwendung geladen werden. Bei all diesen Vorgängen geschehen sehr viele Zugriffsvorgänge auf System-Objekte. Dabei werden Threads blockiert, um auf die angefragten Ressourcen zu warten, und später wieder reaktiviert, wenn die Ressourcen bereitstehen. Dies geschieht für die vielen verschiedenen Threads der Prozesse sehr häufig hintereinander. Aus diesem Grund sind sowohl kurzzeitige Anstiege bei den Kontextwechseln als auch bei den Objekt-Zugriffen zu verzeichnen. Die Objekt-Zugriffe liegen mit Werten von 600/s bis 9.000/s über dem Normalwert. Während des Installationsprozesses wurden jedoch mit 10.000/s bis 60.000/s noch wesentlich mehr Zugriffe gemessen. Wird der Installationsprozess ausgeklammert, stellen jedoch die Startvorgänge die Szenarien mit der größten Objekt-Zugriffsrate dar.

Bei den Startvorgängen zeigt sich dies auch in Bezug auf Lese- und Schreibvorgänge auf realen Dateien auf Datenträgern. Es wurden systemweit etwa bis zu 1.000 Lesezugriffe pro Sekunde und ca. bis zu 600 Schreibzugriffe pro Sekunde gemessen. Für die vielen beim Startvorgang gelesenen Ressourcen braucht es Speicherplatz in Form von neuen Speicherseiten im Adressraum der neu angelegten Prozesse, zudem wird auf die meisten Seiten der virtuellen Prozessadressräume beim Anlaufen der Software zugegriffen. Dies verursacht in einem großen Maße Page-Faults. Für den Code, die Daten und Ressourcen der Executables und der vielen dynamischen Bibliotheken (DLLs) verursachen die Zugriffe auch harte Page-Faults, bei denen die entsprechenden Daten der Seite aus den EXE- und DLL-Dateien geladen werden und in die neu angelegte Seite kopiert werden. Die Messungen bestätigen hier wiederum die Erwartungen. Die Rate der Page-Faults ist bei den Startvorgängen extrem viel höher als bei allen anderen Szenarien. Sie unterscheidet sich jedoch deutlich zwischen den verschiedenen Anwendungen. Dies erscheint logisch, da der Seitenbedarf je nach der Menge der involvierten Bibliotheken und Ressourcen sehr unterschiedlich ausfallen kann. Der Startvorgang des Windows-Explorers verursacht mit ca. 10.000 Page-Faults/s die geringste Rate an Page-Faults im Vergleich aller Start-Szenarien. Die höchste Rate wurde beim Startvorgang der Anwendung LibreOffice-Writer gemessen. Hier verursachten die Prozesse fast 140.000 Page-Faults pro Sekunde. Im Unterschied zum Page-Fault-Verhalten bei den vorherigen Szenarien, die Langzeit-Vorgänge beschreiben, treten daher bei den Startvorgängen harte Page-Faults im deutlich erhöhten Maße auf. Dies ist auf den beschriebenen Sachverhalt des Ladens der Executable-Inhalte (EXE- und DLL-Dateien) zurückzuführen. Beim Start des Windows-Explorers geschehen systemweit ca. 1.300 Hard-Page-Faults pro Sekunde. Der Startvorgang von LibreOffice-Writer verursacht sogar um die 30.000 Hard-Page-Faults pro Sekunde.

Das folgende Diagramm zeigt die schlussendliche Abschätzung der Auftrittsfrequenzen bezüglich der beobachteten Maximal- und Minimal-Werte. Die Frequenzfenster, die für jeden Ereignistyp bestimmt wurden, wurden grob abgeschätzt anhand der gemessenen maximalen und minimalen Werte aller Messungen. Dadurch wird ein Eindruck geliefert, mit welchen Raten und daraus resultierenden Ereignismengen ein Monitoring-System umgehen können muss, wenn es eine der hier herangezogenen Ereignistypen für die System-Überwachung nutzt. Im Diagramm ist auch die Periodendauer abzulesen. Wie erwähnt ist diese rein aus der Frequenz als reziproker Wert errechnet worden und es muss stets berücksichtigt werden, dass die jeweiligen Ereignisse auch in wesentlich kürzeren Abständen auftreten können.

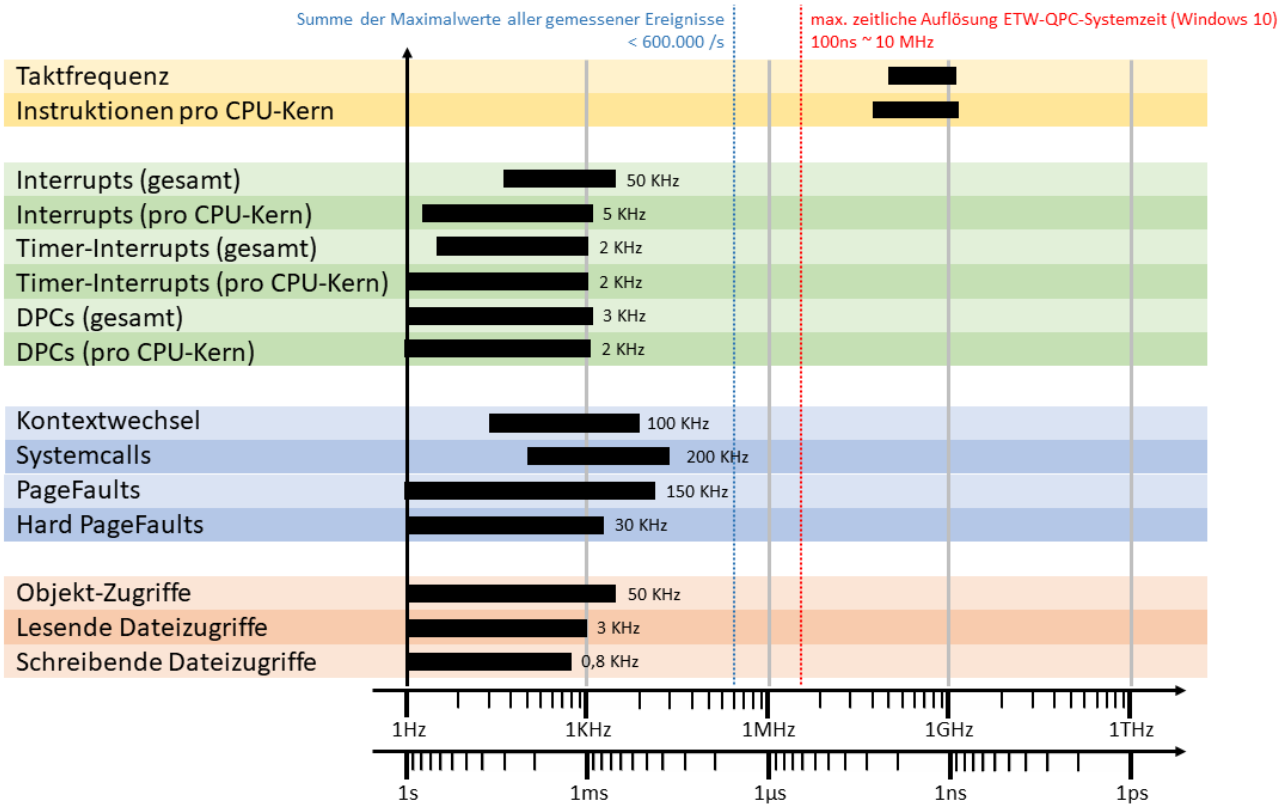

Abbildung 2.2: Abschätzung Ereignis-Auftrittsfrequenzen

3 Event Tracing for Windows (ETW)

Event-Tracing-for-Windows (ETW) ist eine standardmäßig in Windows-Betriebssystemen integrierte ereignisbasierte Ablaufverfolgung. ETW ermöglicht es allgemein beliebige Ereignisse in der auf einem Windows-System laufenden Software, einschließlich der Betriebssystem-Software selbst, aufzuzeichnen. Die Auswertung der aufgezeichneten Ereignisfolge ermöglicht, die technischen Abläufe innerhalb des Gesamtsystems nachzuvollziehen. Dadurch ist ETW ein effektives Werkzeug, um Fehler und Performance-Probleme in Software-Systemen zu erkennen und zu beheben. Darüber hinaus wird ETW von verschiedenen Programmen als Langzeit-Systemüberwachung benutzt. Windows-interne Programme und Mechanismen nutzen ETW, um über bestimmte Ereignisse im Gesamtsystem benachrichtigt zu werden oder spezielle Informationen zu sammeln, wie z.B. Leistungsindikatoren. Beispiele für Programme, die ETW als Informationsquelle nutzen, sind unter anderem der Task-Manager, der Ressourcen-Monitor oder die Windows-eigene Security-Software Windows-Defender. Auch Dritt-Software nutzt ETW-Ereignisse zur Systemüberwachung. So stellt ETW mittlerweile eine der wichtigsten Informationsquellen für EDR-Endpunkt-Software verschiedener Hersteller dar [Yos19, min. 1:26][Gol17, min. 4:15][TKG21a, S. 4][TKG21b, min. 3:25].

Prinzipiell kann jede Software-Komponente das im Windows-Kernel befindliche ETW-System nutzen, um eigene Ereignisse zu signalisieren und diese somit protokollierenden Software-Komponenten zur Verfügung zu stellen. Die Komponente erstellt dazu ETW-Event-Objekte und reicht diese an das ETW-System weiter. Dieses speichert die eingehenden Event-Objekte zwischen und reicht diese an protokollierende Software-Komponenten weiter, die beim ETW-System den Erhalt von Event-Objekten aus bestimmten Quellen angefordert haben [Gol17, min. 6:00][Yos19, min. 4:46]. Für eine erzeugende Instanz muss eine Provider-Klasse im ETW-System installiert sein, für die sich die Komponente, die Event-Objekte erzeugt, anmeldet. Eine Provider-Klasse definiert mehrere Ereignistypen und deren Attribute. Sie liefert für die erzeugende und für die empfangende Komponente die gemeinsame Definition der Ereignistypen, sodass diese auch beim Erhalt korrekt interpretiert werden können. Generell kann jegliche User-Mode-Software über die Windows-API eigene Ereignisse dem ETW-System mitteilen [Mic6a]. Logischerweise können ETW-Ereignisse nicht nur vom Anwendungscode sondern auch von jeglichem Bibliothekscode erzeugt werden. Viele User-Mode-Bibliotheken wie auch die Bibliotheken der Windows-API oder der Dot-NET-Laufzeitumgebung können ETW-Events erzeugen. Beispiele sind die `NTDLL.dll`, `KernelBase.dll`, `Shell32.dll`, `AdvAPI32.dll` und `Setup.dll`. Auch die Windows Bibliotheken für Sicherheit und Verschlüsselung wie z.B. `WinTrust.dll`, `Amsi.dll`, `WsCapi.dll`, `Rsaenh.dll`, `Ncrypt.dll`, `Bcrypt.dll`, `Crypt32.dll`, `FirewallApi.dll` und `FwpucInt.dll` oder für Netzwerk wie z.B. `UrlMon.dll`, `WinHTTP.dll`, `WebIO.dll`, `Wbemcomn.dll`, `lertUtil.dll` und

`DnsAPI.dll` erzeugen Event-Objekte für das ETW-System [TKG21a, S. 5][TKG21b, min. 5:19]. Nicht nur Microsoft selbst signalisiert interne Software-Ereignisse mittels ETW, auch etliche Software von Dritt-Anbietern nutzt die Windows-Funktionalität. Neben einer Reihe von Anwendungsprogrammen wie z.B. die Browser Chrome, Firefox und Edge oder die Programme der Microsoft Office-Suite erzeugen auch Befehlsinterpreter für die Ausführung von Skripten wie CMD oder die PowerShell ETW-Ereignisse. Auch die Windows-internen Anwendungsprogramme wie z.B. Notepad, Wordpad, Paint und der Windows-Explorer sowie die Security-Prozesse NisSrv, MsMpEng und Sgrm-Broker signalisieren genauso wie die Windows-System-Prozesse Smss, Lsass, WinLogon, Wininit oder die NT-Services ihre internen Ereignisse über ETW [TKG21a, S. 5]. Neben User-Mode-Komponenten können auch Kernel-Komponenten Ereignisse über ETW signalisieren. Dazu stellt Microsoft Treiberentwicklern eine Kernel-API für die Nutzung von ETW zur Verfügung. Microsoft selbst erzeugt im Code der eigenen Kernel-Komponenten an vielen Stellen ETW-Event-Objekte. Somit lässt sich durch ETW nicht nur das Verhalten der User-Mode-Anwendungen protokollieren sondern auch das der Kernel-Komponenten. Im Windows-Kernel selbst (`ntoskrnl.exe`) erzeugen alle Basismechanismen wie Prozess-Manager (PS), Objekt-Manager (OB), Config-Manager (CM), IO-Manager (IO) usw. ETW-Ereignisse. Auch der für die grafische Oberfläche und die Desktop-Technologien zuständige Win32-Kernel (`win32k.sys`) erzeugt ebenfalls ETW-Events. Die Windows-internen Kernel-Treiber für Security, Encryption, Netzwerk und Systemgeräte wie z.B. `Cng.sys`, `FileCrypt.sys`, `KSecdd.sys`, `MsSecFlt.sys`, `Tbs.sys`, `NTFS.sys`, `CimFS.sys`, `WciFS.sys`, `VwifiFlt.sys`, `CldFlt.sys`, `BindFlt.sys`, `HTTP.sys`, `TcpIp.sys`, `NetIO.sys`, `Partmgr.sys`, `VloMgr.sys`, `NDIS.sys`, `Disk.sys`, `CDrom.sys`, `ClassPNP.sys` erzeugen ebenfalls ETW-Events [TKG21a, S. 5]. Die hier aufgezählten Komponenten stellen nur einen Teil aller auf einem Windows-System vorhandener Komponenten dar, die ETW für die Signalisierung von Ereignissen nutzen. Für jede dieser Komponenten muss eine entsprechende ETW-Provider-Klasse auf dem System installiert sein. In einem frisch installierten Windows 11 sind über 1000 verschiedene Provider-Klassen vorinstalliert, die über 50.000 verschiedene Ereignistypen beschreiben [TKG21b, min. 5:19][Gol17, min. 8:27][Yos19, min. 18:00].

Neben der Möglichkeit als ein Analyse-Prozess die im ETW-System eingehenden Event-Objekte direkt aus dem ETW-System im Kernel zu beziehen, bietet ETW auch die Funktionalität, die eingehenden Event-Objekt selbstständig aus dem Kernel heraus in Dateien auf Datenträgern zu protokollieren. Diese ETL-Dateien können dann zu einem späteren Zeitpunkt von der Analysesoftware mit Hilfe der ETW-API eingelesen werden.

Eine häufig erwähnte Eigenschaft des ETW-Systems ist die performante Implementierung im Windows-Kernel mit einem niedrigen Overhead bezüglich der Ereigniserzeugung, -Zwischenspeicherung und Bereitstellung. So sollen tausende Ereignisse pro Sekunde erzeugt und zwischengespeichert werden können, ohne dass es zu einer, für den Benutzer

erkennbaren, Verzögerung kommt. Microsoft gibt in seinen Dokumentationen an, dass ETW in der Lage ist, 100.000 eingehende Ereignisse pro Sekunde zu verarbeiten. Dabei wird der Overhead durch die Angabe beschrieben, dass 10.000 Ereignisse pro Sekunde etwa 5% zusätzliche CPU-Auslastung verursachen [Reu20, S. 53][Gol17, min. 1:07][Gol17, min. 4:15].

Event-Tracing-for-Windows (ETW) wurde im Jahr 2000 mit der Windows NT Version Windows 2000 (NT 5.0) erstmals eingeführt und seit dem stetig erweitert. Anfänglich wurde ETW zu seinem primären Nutzen als Debugging- und Profiling-Werkzeug in der Software-Entwicklung, im speziellen der Treiber-Entwicklung, eingesetzt. Vor allem jedoch sollte ETW das Debugging und Profiling bei den Microsoft-internen Entwicklungsarbeiten am Windows-Kernel verbessern. Aus diesem Grund enthält ETW eine Menge an undokumentierten Funktionen und Provider-Klassen, die Microsoft intern nutzt, jedoch nicht über die verschiedenen ETW-APIs Drittparteien zur Verfügung stellt. Erst seit ca. 2012 gelangte ETW immer mehr in der Fokus der IT-Sicherheits-Industrie. Dort wird die Ablaufverfolgung als Sensorik für die Systemüberwachung eingesetzt. ETW findet sich als Informationsquelle in Werkzeugen für die dynamische Malware-Analyse sowie in EDR-Endpunkt-Sensorik.

3.1 ETW im Sicherheitskontext

Im Jahr 2012 präsentierte Mark Russinovich in der Präsentation „Malware Hunting with the Sysinternals Tools“ erstmalig die Verwendung von ETW im Kontext einer Endpunkt-Sensorik [Rus12]. Er benutzte ETW unter anderem in seinen Monitoring-Tools Sysmon und Procmon für die Erkennung von Ereignissen des Kernel-internen Netzwerk-Stapels (TCP, UDP, IP). Dies führte in den darauf folgenden Jahren dazu, dass IT-Sicherheitsforscher der Technologie immer größere Aufmerksamkeit schenkten und auch IT-Sicherheitsfirmen begannen, Softwarelösungen zu entwickeln, die ETW als Informationsquelle benutzen. Mittlerweile verwenden alle führenden EDR-Anbieter in ihren Endpunkt-Sensoren (EDR-Agents) auch neben anderen Verfahren ETW für die Erkennung bestimmter Systemereignisse [TKG21a, S. 6][TKG21b, min. 5:47]. In den letzten Jahren entwickelten sich so viele Ideen, Ansätze, Prototypen und Lösungen rund um das Thema der ETW-Nutzung für die Angriffserkennung. Darunter waren auch Konzepte für offene EDR-Lösungen, sodass neben den kommerziellen Lösungen der großen EDR-Anbieter wie beispielsweise CrowdStrike, Microsoft und Symantec auch einige Open-Source-Software-Lösungen entstanden [TKG21a, S. 7][TKG21b, min. 7:40]. Begleitet wurde diese Entwicklung durch Forschungsaktivitäten im IT-Sicherheitsbereich, die das Thema ETW adressierten. So können rund um die 100 Forschungsprojekte gefunden werden, die sich mit dem Thema der ETW-Nutzung im Kontext der IT-Sicherheit

beschäftigen [TKG21a, S. 8][TKG21b, min. 7:60]. Diese Aktivitäten brachten einige bedeutende Veröffentlichungen hervor [TKG21a, S. 9][TKG21b, min. 10:30]. Zu nennen sind hier Zac Brown mit der Publikation „Hidden Treasure: Detecting Intrusion with ETW“, welche der Frage nachgeht, wie ETW in der IT-Forensik eingesetzt werden kann [Bro17], oder Matt Hastings und Dave Hull mit „Tracing Adversaries: Detecting Attacks with ETW“[HH17] und Joe Desimone mit „Hunting for Memory-Resident Malware“ [Des17]. Zudem stellt ETW einen wichtigen Bestandteil in den drei DARPA-Projekten WLLSMDC, MARPLE und APTShield dar. „Windows Low-Level System Monitoring Data Collection“ (WLLSMDC) liefert Erkenntnisse hinsichtlich der Sichtbarkeit von ETW im Bezug auf bösartige Anwendungsfälle wie Exploitationen oder dem Download von Schadsoftware. Der Fokus liegt dabei auf der ETW-System-Provider-Klasse „NT Kernel Trace“ [DAR]. Das MAPLE Projekt forscht an dem Thema EDR-Lösungen für die Erkennung komplexer APT-Kampagnen. ETW spielt dabei eine wichtige Rolle für die Sensorik [Cen]. APTShield fokussiert sich auf die Erkennung von Schadsoftware, die versucht, Vorgänge auf dem infizierten Rechner zu beobachten, die geheime Informationen böswillig auszuleiten. Dies beinhaltet unter anderem die Erkennung von Remote-Zugriffen, Registry-Manipulationen, Key-Logger-Funktionalitäten, das Aufzeichnen von Bildschirminhalten, sowie verdeckte Audio- und Video-Aufnahmen durch eine Webcam. Auch für diese Erkennung wird im APTShield-Projekt ETW eingesetzt [Che]. Neben der DARPA (US Verteidigungsministerium) betreiben auch MITRE (US Forschungs-Infrastruktur-Betrieb) und das deutsche BSI (Bundesbehörde für IT-Sicherheit) ETW-Projekte. MITRE arbeitet an einem auf ETW basierenden Security-Sensor, der ähnliche Informationen wie Sysmon liefern soll (Process-Erstellung und -Injection, Datei und Registry-Manipulationen, Löschvorgänge usw.) [SBK+]. In Deutschland untersucht die Heidelberger IT-Firma ERNW zusammen mit dem Bundesamt für Sicherheit in der Informationstechnik (BSI) im Projekt SISyPHuS wie sich ETW-basierende Protokollierungen hinsichtlich einer Angriffserkennung analysieren lassen können [BfSidI].

3.2 ETW vs. PCW

Manche Quellen bezeichnen Event-Tracing-for-Windows (ETW) als Nachfolger von Performance-Counters-for-Windows (PCW) [Wik18]. Zwar ist der primäre Anwendungszweck zwischen ETW und den Leistungszählern sehr ähnlich, so unterscheiden sich die Konzepte dennoch. ETW zeichnet Folgen von Einzelereignissen auf und stellt diese als Live-Ereignisstrom oder persistierte Protokollierung bereit. Jedes Ereignis-Objekt kann dabei je nach Ereignistyp beliebige Metainformationen enthalten, die das einzelne Ereignis beschreiben. Dahingegen sind die Windows-Performance-Counters lediglich Zählvariablen, die automatisch erhöht oder anderweitig gesetzt werden, wenn bestimmte Ereignisse geschehen. Dadurch kann ermittelt werden, wie viele Ereignisse eines bestimmten

Typs in einer bestimmten Zeit geschehen sind. Detaillierte Informationen, wann einzelne Ereignisse aufgetreten sind sowie deren Attribute, kann PCW nicht liefern [Yos19, min. 9:20].

Es gibt Anwendungsfälle in denen ist es notwendig, den Ablauf auf Ereignis-Granularität im Detail zu verfolgen oder detaillierte Informationen einzelner Ereignisse zu erhalten. In diesem Fall ist ETW die passende Technologie. Reicht dem Anwendungsfall jedoch ein reines Leistungsmonitoring auf Basis von Leistungsindikatorwerten (Auslastung pro Sek., Interrupts pro Sek., Systemcalls pro Sek. usw.) aus, wäre der Einsatz von ETW zwar möglich jedoch überdimensioniert. PCW reicht in diesem Fall aus, die benötigten Leistungsindikatorwerten zur Verfügung zu stellen.

Da beide Systeme in Windows parallel zueinander existieren und verwendet werden können, sollte ETW eher nicht als Nachfolger von Performance-Counters-for-Windows (PCW) sondern als eine separate, ergänzende Technologie angesehen werden.

3.3 Klassisches ETW und modernes ETW

Event-Tracing-for-Windows (ETW) wurde mit der Einführung von Windows Vista einer großen Erweiterung unterzogen. Es wurde unter anderem ein neuer Provider-Klassen-Typ eingeführt, der auf Instrumentierungsmanifesten im XML-Format basiert. Für die Verwendung dieser neuen Manifest-basierten Provider-Klassen in ETW-Sessions gelten wesentlich weniger Limitierungen bezüglich der gleichzeitigen Nutzung in anderen Sessions. Für die Nutzung der neuen Manifest-basierten Provider-Klassen wurde eine neue ETW-API hinzugefügt. In der Windows-Dokumentation werden die vor Windows Vista existierenden Strukturen und die alte API als klassisches ETW (classic ETW) bezeichnet, um diese von der neueren, moderneren ETW-Variante zu unterscheiden. Das klassische ETW basiert auf WMI-Technologien wie zum Beispiel MOF-Klassen und -Objekten, wie sie auch in WMI genutzt werden. Das neuere, moderne ETW verwendet zumindest auf Nutzungsebene keine WMI-Technologien mehr [Mic6a][AIRS21, S. 499ff]. Entwicklern wird empfohlen zukünftig die neuen Manifest-basierten Provider-Klassen zu verwenden. Mit Windows 8 und Windows 10 wurde ein Großteil der nur im klassischen ETW bestehenden MOF-Provider-Klassen zusätzlich als moderne Manifest-basierte Provider-Klassen hinzugefügt [MicBa].

3.4 WMI-Ursprünge

Ursprünglich ist ETW zusammen mit den Windows-Management-Instrumentarien (WMI) in Windows 2000 eingeführt worden und war ursprünglich mit WMI gekoppelt. So besitzt WMI eine eigene Ablaufverfolgung, welche den technischen Ursprung von ETW bildet. Seit Windows XP ist ETW aus Nutzungssicht eigenständig und unabhängig von WMI zu benutzen. Seit Windows Vista wird empfohlen, für eine Ablaufverfolgung in WMI statt der WMI-Ablaufverfolgung ETW zu nutzen. Nicht desto trotz ist die Verwandtschaft von WMI und ETW noch an sehr vielen Stellen zu erkennen. So basiert vor allem das klassische ETW auf WMI-Datenstrukturen und -Verfahren. ETW-Provider-Klassen werden in gleicher Weise wie WMI-Klassen durch MOF-Klassen (Managed Object Format) beschrieben. Die Serialisierung und Kodierung der Event-Daten benutzt die gleichen Algorithmen wie das WMI-System. Darüber hinaus ist die Verwandtschaft an vielen Bezeichnern von Funktionen, Datenstrukturen und Registry-Schlüsseln der ETW-API zu erkennen, die noch die Abkürzung WMI in ihren Zeichenketten tragen. Um die internen, nicht dokumentierten Abläufe und Datenstrukturen des ETW-Systems zu verstehen, kann es daher hilfreich sein, die Funktionsweise und Strukturen von WMI zu verstehen, da ETW intern ähnlich aufgebaut ist und Teile des WMI-Systems nutzt.

3.5 Eigenschaften

In diesem Abschnitt wird auf die drei Basiseigenschaften „Ausführungskontext“, „Auftrittszeit“ und „Abstraktionsebene“, die im ersten Kapitel allgemein eingeführt und erläutert wurden, Bezug genommen und geprüft, wie diese bei Event-Tracing-for-Windows (ETW) umgesetzt sind.

3.5.1 Ausführungskontext

In jedem ETW-Event-Objekt ist der Ausführungskontext des Ereignisses gespeichert. Im allgemeinen Header eines jeden ETW-Event-Objektes wird im Feld `ThreadId` die ID-Nummer des Threads hinterlegt, in dessen Ausführungskontext das Ereignis aufgetreten ist. Im Feld `ProzessId` wird die ID-Nummer des Prozesses gespeichert, dem der Thread, in dessen Kontext das Ereignis aufgetreten ist, angehört. Neben dem Header besitzt jedes ETW-Event-Objekt noch eine allgemeine Struktur, die den Puffer-Kontext spezifiziert. Aus dieser kann indirekt der Prozessor-Kontext ermittelt werden. Die Struktur `ETW_BUFFER_CONTEXT` eines ETW-Event-Objektes enthält das Feld `ProzessorIndex`, in welchem die Nummer des logischen Rechenkerns gespeichert wird, in dessen Kontext das Ereignis aufgetreten ist und das Event-Objekt erzeugt wurde [Mic5a].

Der Ausführungskontext wird zu dem Zeitpunkt bestimmt, an dem das ETW-Event in den Puffer der ETW-Session eingetragen wird. Dieser Zeitpunkt liegt einige Befehle nachdem das eigentliche Ereignis, welches durch das Event-Objekt signalisiert wird, aufgetreten ist. Daher entspricht der Ausführungskontext der im Event-Objekt beschrieben wird, in der Regel dem Ausführungskontext des originalen Ereignisses. In seltenen Fällen kann es vorkommen, dass der Ausführungskontext zum Zeitpunkt des Einfügens des ETW-Event-Objektes in ETW-Sessions bereits anders ist als einige Instruktionen zuvor, als das zu signalisierende Ereignis auftrat. Dieses Problem betrifft jedoch nur die Information bezüglich des Prozessor-Kontextes. Geschieht zwischen dem Auftreten des originalen Ereignisses und dem Erstellen des ETW-Events ein Kontextwechsel, ist es möglich, dass der im Event-Objekt eingetragene logische Rechenkern nicht der logischen CPU entspricht, in dessen Kontext das Ereignis aufgetreten ist. Die Informationen bezüglich des Threads und Prozesses im Event-Objekt sind jedoch jederzeit korrekt und entsprechen dem Ausführungskontext des realen Ereignisses.

3.5.2 Auftrittszeit

Jedes ETW-Event-Objekt weist in seinem allgemeinen Header das Attribut `TimeStamp` auf. Dieses enthält einen Zeitstempel, der spezifizieren soll, wann das assoziierte Ereignis aufgetreten ist. Der allgemeine Header eines ETW-Event-Objekt enthält zusätzlich zum `TimeStamp`-Feld die drei Felder `ProcessorTime`, `UserTime` und `KernelTime`. Alle drei Attribute werden nur im Kontext von privaten ETW-Sessions genutzt und sind somit für diesen Abschnitt nicht relevant. Alle in diesem Abschnitt beschriebenen Zeitstempel-Werte werden in das Feld `TimeStamp` des Headers eingetragen [Mic5a].

ETW benutzt für alle Event-Objekte derselben ETW-Session denselben Zeitgeber. Dieser kann für jede Session frei konfiguriert werden. Im ETW-System stehen drei verschiedene Zeitquellen zur Auswahl. Verschiedene parallel betriebene Sessions können verschiedene Zeitgeber nutzen. Da innerhalb einer Session jedoch für alle Ereignisse dieselbe Zeitquelle benutzt wird, sind die Event-Objekte einer Session zeitlich vergleichbar und besitzen somit zumindest bezogen auf den eingetragenen Zeitstempel eine chronologische Ordnung. Letztere ergibt sich nur auf Basis des Zeitstempels. Ist dieser nicht korrekt, ist die Reihenfolge, in welcher der ETW-Consumer die Event-Objekte empfängt, eventuell falsch bzw. entspricht nicht genau der Reihenfolge, in der die originalen Ereignisse aufgetreten sind [Mic6b].

Der Wert des Zeitstempels wird bei jedem Einfügevorgang eines Event-Objektes in den Puffer einer Session, die dieses Ereignis abonniert hat, von dem, in der Session spezifizierten, Zeitgeber abgefragt und in das Timestamp-Feld im Header des jeweiligen Eintrages gespeichert [EMY19]. Dabei kann das eine Event-Objekt auch als Kopie mehrmals

in verschiedene Puffer verschiedener ETW-Sessions eingetragen werden. Dies ist dann der Fall, wenn der Event-Typ von mehreren ETW-Session gleichzeitig abonniert wurde. Dementsprechend kann für jeden Eintrag des Event-Objektes ein anderer Zeitstempel eingetragen werden, wenn für die verschiedenen ETW-Session verschiedene Zeitquellen spezifiziert sind. Da der Zeitstempel erst vor dem Einfügen des Event-Objektes in den Session-Buffer im Kernel ermittelt wird, besteht eine gewisse Latenz zwischen dem Zeitpunkt, der in das Event-Objekt eingetragen wird und dem Zeitpunkt, an dem das originale Ereignis aufgetreten ist.

Die drei zur Auswahl stehenden Zeitquellen, die für eine ETW-Session spezifiziert werden können, sind der Query-Performance-Counter (QPC), die Windows-Systemzeit und der CPU-Zykluszähler [Mic5b].

3.5.2.1 Query-Performance-Counter (QPC)

Der Query-Performance-Counter (QPC) ist ein vom Windows-Kernel verwalteter Zeitgeber, der im Optimalfall sehr hohe Präzision aufweist. Der vom Betriebssystem-Kernel gelieferte Zeitwert ist ein 64-Bit-breiter Zeitstempel mit variabler zeitlicher Auflösung. Er kann je nach Hardware-Plattform auf verschiedenen Zählerschaltungen basieren, von deren Genauigkeit die Auflösung des QPC abhängt. Windows stellt Software, die QPC-Zeitstempel verwenden und interpretieren müssen, wie z.B. den ETW-Consumern, den Frequenzwert des verwendeten Hardware-Zählers bereit, sodass die Auflösung des QPC-Zeitstempels ermittelt werden kann. Der Windows-Kernel sorgt dafür, dass der QPC-Zeitstempel beim Systemstart mit Null initialisiert wird. Sein Wert ist unabhängig vom CPU-Kern, von welchem aus der Wert abgefragt wird und er ist über die gesamte Betriebszeit eines Computersystems stabil. Microsoft gibt an, dass es etwa nach 100 Jahren Betriebszeit ohne Neustart zu einem Überlauf des Zählers kommen könnte [Mic4a][Mic5b].

Besitzen alle logischen CPU-Kerne einen von der CPU-Kern-Frequenz unabhängigen Zeitzähler (TSC), der unter allen Rechenkernen synchronisiert ist, wird dieser TSC-Zähler als Zeitquelle für QPC verwendet. Die Frequenz des TSCs variiert zwischen den CPU-Modellen. Bei einer Beispielfrequenz des TSC von 3 GHz würde die Auflösung des QPC-Zeitstempels 0,333 ns betragen. Bei PC-Prozessoren (IA32/AMD64) erfolgt der Zugriff auf den TSC über den Maschinenbefehl `RDTSC`. Diese Instruktion wird von der QPC-Bibliothek verwendet. Wird die QPC-API aus dem User-Mode genutzt, führt die dahinter liegende Bibliothek den `RDTSC`-Befehl aus dem User-Mode aus, ohne einen Systemcall zu tätigen. Dies hat den großen Vorteil, dass die Verwendung von auf TSC basierendem QPC eine sehr geringe Latenz von nur ca. 25 ns aufweist. Diese Latenz gibt somit die maximale zeitliche Schranke bezüglich der Genauigkeit an [Mic4a][Mic5b].

Erfüllen die TSC-Zähler im Prozessor nicht die Anforderungen (alle TSCs synchronisiert und unabhängig der Kern-Frequenzen), wird für QPC eine alternative Zeitquelle verwendet. Besitzt die Hardware-Plattform eine HPET-Schaltung (High Precision Event Timer), wird diese verwendet, ansonsten wird der Power-Management-Timer des ACPI-Systems (ACPI-PM-Timer) benutzt. Beide IO-Geräte müssen über Treiber angesprochen werden und es muss ein Datentransport über Interconnects und Bussysteme stattfinden. Dies führt zu einer variablen Latenz von ca. 900 ns. Basiert QPC nicht auf TSC ist logischerweise immer ein Systemcall notwendig, wenn QPC aus dem User-Mode benutzt wird. Zudem ist die Auflösung der HPET- oder ACPI-PM-Timer-Schaltung meist geringer als die des TSC. Sie liegt trotzdem in der Regel aber auch im Nanosekundenbereich. Aufgrund der nicht deterministischen, variablen Zugriffslatenz ist die Genauigkeit mit etwa einer Mikrosekunde anzugeben [Mic4a][Mic5b].

Da die QPC-Abfrage im Kontext von ETW vor dem Einfügen des Event-Objektes in den Session-Buffer stattfindet [EMY19], geschieht diese in der Regel immer im Kernel-Mode (außer bei privaten ETW-Sessions), wodurch der Unterschied, ob für die QPC-Abfrage aus dem User-Mode ein Systemcall notwendig ist oder nicht, für den ETW-Kontext unerheblich ist.

QPC eignet sich als ETW-Zeitgeber, wenn es ausreicht, dass die Zeitpunkte in den Event-Objekten relative Zeitpunkte, gemessen als zeitlicher Abstand zum Systemstart, sind. Sollen nur Ereignisse innerhalb einer Aufzeichnung in Beziehung gesetzt werden, reicht dies meist aus. Ist eine absolute Zeitinformation nötig, um Event-Daten über verschiedene Aufzeichnungen hinweg in eine zeitliche Beziehung zueinander zu setzen, kann QPC nicht benutzt werden. Zudem kann QPC als Zeitquelle für ETW eine gute Wahl sein, wenn hochfrequente Systemereignisse protokolliert werden sollen, da QPC prinzipiell hohe zeitliche Auflösungen erlaubt. Sollte QPC nicht auf TSC basieren, kann die zeitliche Auflösung jedoch zu gering sein, um Systemereignisse korrekt zu erfassen und in der richtigen Reihenfolge zu protokollieren. Auch bei Verwendung von TSC kann durch die Latenz von QPC die Reihenfolge der Event-Objekte in seltenen Fällen von der originalen Reihenfolge der Ereignisse abweichen. Dies ist dadurch begründet, dass die ETW-Event-Erzeugung und Speicherung auf verschiedenen CPU-Kernen parallel erfolgen kann und aus Gründen der Performance an dieser Stelle auf Locks verzichtet wurde (einen Puffer-Kontexte für jeden CPU-Kern). Die Prozessoren in heutigen modernen PCs verwenden TSCs, welche die oben genannten Bedingungen erfüllen, sodass in den meisten Fällen davon ausgegangen werden kann, dass QPC auf TSC basiert und eine sehr hohe zeitliche Auflösung bietet [Mic4a][Mic5b][Mic5a].

Die Frequenz des QPC-Zeitgebers schreibt das ETW-System in das Feld `PerfFreq` der `TRACE_LOGFILE_HEADER`-Struktur, welche die ETW-Session spezifiziert. Dadurch ist ein

möglicher ETW-Consumer in der Lage, den Zeitstempel des ETW-Ereignisses korrekt zu interpretieren [Mic5b][Mic5a].

Query-Performance-Counter (QPC)

basiert auf:	TSC / HPET / ACPI-PM-Timer (meist TSC)
Typ:	Hardware-Zähler
Realzeit:	nein
Format:	64 Bit Zeitstempel
Auflösung Zählvariable:	hardwareabhängig
zeitliche Auflösung:	hardwareabhängig ($<$ 0,3 ns möglich)
zeitliche Genauigkeit:	hardwareabhängig (maximal ca. +/- 30 ns)
Nullpunkt:	Systemstart
Stabilität:	stabil

Tabelle 3.1: Steckbrief: Query-Performance-Counter (QPC)

3.5.2.2 Windows-Systemzeit

Die Windows-Systemzeit ist eine im Windows-Kernel verwaltete Zeitquelle und bildet eine Kombination aus der Windows-Interruptzeit und der Realzeit. Sie basiert auf einer Software-Uhr mit einer variablen zeitlichen Auflösung, die in der Regel maximal 500 µs betragen kann [Mic5b][Mic4b][Mic4c].

Windows konfiguriert den System-Timer, eine Hardware-Schaltung, die periodisch Interrupts auslöst, sodass regelmäßig Timer-Interrupts auf einem logischen CPU-Kern ausgelöst werden. In modernen PCs existieren verschiedene Hardware-Schaltungen, die als System-Timer dienen können (PIT, APIC-Timer, RTC-Timer, ACPI-Power-Management-Timer, HPET). Die verwendete Schaltung hängt dabei von Hardware-Plattform und der Firmware-Konfiguration (BIOS Setup) ab. In der Regel wird die PIT-Schaltung (Programmable Interval Timer) verwendet und auf eine Frequenz zwischen 66 Hz und 2 KHz eingestellt, was einer Periodendauer zwischen 15,625 ms und 500 µs entspricht. Windows variiert während des Betriebs die Frequenz des PIT nach Auslastung und Anforderung stufenweise zwischen den zwei Grenzwerten. Die Interrupt-Service-Routine (ISR) des Kernels für den Timer-Interrupt (Timer-ISR) erhöht einen 64-Bit-Zähler mit einem Wert, der der aktuell eingestellten Periodendauer in 100 ns entspricht. Die Zählvariable wird beim Booten vor dem Start des Systemtimers mit Null initialisiert. Der Wert des Zählers wird bei jedem eingegangenen Timer-Interrupt um die aktuell eingestellte Periodendauer des Timer-Interrupts erhöht und gibt somit die Zeit seit dem Systemstart an. Diese auf den Timer-Interrupts basierende Zeit wird als Windows-Interruptzeit bezeichnet. Sie ist ein Zeitstempel mit der Werte-Auflösung 100 ns und einer variablen zeitlichen Auflösung zwischen 15,625 ms und 500 µs [AIRS21, S. 66ff][Han19]. Die zeitliche Auflösung

wird durch die Hardware-Abstraktionsschicht (`HAL.dll`) bestimmt und ist somit abhängig von der Plattform und verwendeten Hardware. Sie ist hier für moderne PCs unter der Verwendung des PIT angegeben. Wird eine andere Timer-Schaltung verwendet oder Windows auf einer anderen Plattform ausgeführt, kann die zeitliche Auflösung anders ausfallen.

Beim Booten wird eine zweite Zählvariable mit dem Zeitstempel der Echtzeit, bestehend aus Datum und Uhrzeit, im selben Format (mit 100 ns Auflösung) geladen. Den Nullpunkt des Zeitstempels stellt der Zeitpunkt 1. Januar 1601 0:00:00 Uhr UTC dar. Die Echtzeit-Werte werden aus der Hardware-Echtzeituhr (RTC) gelesen. Diese besitzen eine deutlich geringere Auflösung als 100 ns, sodass die Stellen für Mikro- und Nanosekunden mit Null initialisiert werden. Zudem ist die Hardware-Echtzeituhr unabhängig ihrer begrenzten zeitlichen Auflösung auch in der Ganggenauigkeit nicht besonders präzise, sodass die Windows-Systemzeit ohne eine externe Synchronisierung nur bedingt für absolute Zeitmessungen geeignet ist. Auch dieser zweite Zähler wird mit jedem eingehenden Timer-Interrupt, um die entsprechende eingestellte Periodendauer des Timers in 100 ns erhöht. Dieser Wert wird als Windows-Systemzeit bezeichnet. Die Windows-Systemzeit ist somit die Anzahl an 100 ns Zeiträumen seit dem 1. Januar 1601 0:00:00 Uhr UTC mit einer variablen zeitlichen Auflösung zwischen 15,625 ms und 500 µs (auf PCs bei Verwendung des PIT) [Han19][Mic4b][AIRS21, S. 66ff].

Anders als die Windows-Interruptzeit kann die Windows-Systemzeit im Betrieb angepasst werden. Solche Änderungen werden vom Betriebssystem-Kernel auf Änderungsanfragen durch Prozesse vorgenommen. Dies kann z.B. bei der Synchronisation der Systemzeit mit einer externen Zeitquelle im Internet über das Network-Time-Protocol (NTP) geschehen [Mic5b][Mic4b].

Für die Verwendung der Windows-Systemzeit im Kontext von ETW ist Folgendes zu berücksichtigen. Änderungen der Systemzeit z.B. durch NTP-Synchronisation während einer Ereignisaufzeichnung kann zu Sprüngen in der Zeitbasis der Event-Objekte führen. Dies kann im schlechtesten Fall dazu führen, dass die Reihenfolge der Event-Objekte fehlerhaft protokolliert wird. Zudem kann die zeitliche Auflösung von maximal 500 µs für die Protokollierung hochfrequenten Systemvorgänge zu gering sein. In diesem Fall kann es dazu kommen, dass mehrere Event-Objekte mit dem gleichen Zeitstempel erstellt werden. Die Reihenfolge dieser Event-Objekte kann dadurch im Event-Log anders sein, als die originale Reihenfolge der aufgetretenen Ereignisse [Mic5b][Mic4b].

Die Frequenz des Systemtimer-Interrupts schreibt das ETW-System in das Feld `TimerResolution` der `TRACE_LOGFILE_HEADER`-Struktur, welche die ETW-Session spezifiziert. Dadurch wird es dem ETW-Consumer möglich den Zeitstempel des ETW-Events korrekt zu interpretieren [Mic5b][Mic5a].

Windows-Systemzeit (< Windows 10)

basiert auf:	Systemtimer-Interrupts (meist PIT)
Typ:	Software-Uhr
Realzeit:	ja
Format:	64 Bit Zeitstempel
Auflösung Zählvariable:	100 ns
zeitliche Auflösung:	variable (Plattform-abhängig) (maximal meist 500 µs)
zeitliche Genauigkeit:	variable (Plattform-abhängig) (maximal meist 500 µs)
Nullpunkt:	1. Januar 1601 0:00:00 Uhr UTC
Stabilität:	Sprünge bei Änderungen der Systemzeit

Tabelle 3.2: Steckbrief: Windows-Systemzeit (< Windows 10)

Seit Windows 10 wird nicht die Windows-Interruptzeit, sondern der Query-Performance-Counter (QPC) als Basis für die Windows-Systemzeit verwendet. Dies vergrößert unter Umständen die zeitliche Auflösung deutlich, in Abhängigkeit der für QPC verwendeten Hardware. Bei Windows 10 PCs (moderne 64-Bit CPUs) ist davon auszugehen, dass QPC fast immer auf TSC basiert. Die Umstellung der Windows-Systemzeit auf QPC-Basis ermöglicht es auch hochfrequente Systemereignisse mit der Zeitquelle Windows-Systemzeit und somit einem Realzeit-Bezug zu protokollieren. Anders als bei QPC als ETW-Zeitgeber besitzen die Zeitstempel bei der Verwendung der Windows-Systemzeit einen absoluten Zeitpunkt bezogen auf den 1. Januar 1601 0:00:00 Uhr UTC. Somit ist es möglich die Event-Daten aus verschiedenen Aufzeichnungen zeitlich miteinander zu vergleichen und zu sortieren. Der Nachteil ist jedoch, dass bei Änderungen der Systemzeit während einer Aufzeichnung zeitliche Inkonsistenzen entstehen [Mic5b].

Die Frequenz des QPC-Zeitgebers (TSC) schreibt das ETW-System in das Feld `PerfFreq` der `TRACE_LOGFILE_HEADER`-Struktur, welche die ETW-Session spezifiziert. Dadurch wird es dem ETW-Consumer möglich den Zeitstempel des ETW-Ereignisses korrekt zu interpretieren [Mic5b][Mic5a].

Windows-Systemzeit (>= Windows 10)

basiert auf:	TSC / HPET / ACPI-PM-Timer (meist TSC)
Typ:	Hardware-Zähler
Realzeit:	ja
Format:	64 Bit Zeitstempel
Auflösung Zählvariable:	100 ns
zeitliche Auflösung:	hardwareabhängig (< 0,3 ns möglich)
zeitliche Genauigkeit:	hardwareabhängig (maximal ca. 30 ns)
Nullpunkt:	1. Januar 1601 0:00:00 Uhr UTC
Stabilität:	Sprünge bei Änderungen der Systemzeit

Tabelle 3.3: Steckbrief: Windows-Systemzeit (>= Windows 10)

3.5.2.3 CPU-Zykluszähler

Neben QPC und der Windows-Systemzeit kann auch der direkte Wert des CPU-Zykluszählers (TSC-Register) als Zeitgeber für eine ETW-Session konfiguriert werden. In diesem Fall verwendet ETW als Wert für den Zeitstempel eines ETW-Event-Objektes direkt den Wert des TSC-Registers des logischen Rechenkerns. Auf Hardware-Plattformen deren Prozessor keinen Zykluszähler unterstützt, wird stattdessen der Zeitstempel der Windows-Systemzeit benutzt. Ist der CPU-Zykluszähler als Zeitquelle für eine ETW-Session konfiguriert, liest das ETW-System auf PCs direkt den TSC-Wert über den Maschinenbefehl `RDTSC` aus. Es wird nicht sichergestellt, ob die TSC-Zähler der einzelnen CPU-Kerne synchronisiert sind, korrekt initialisiert wurden und unabhängig von den Frequenzen den Rechenkerns betrieben werden. QPC stellt im Gegensatz dazu die aufgeführten Bedingungen sicher. Auf modernen PCs kann aber davon ausgegangen werden, dass die Bedingungen in den meisten Fällen erfüllt sind [Mic5b][Mic5a].

Der CPU-Zykluszähler ist die ETW-Zeitquelle mit der höchsten zeitlichen Auflösung und gleichzeitig der geringsten Zugriffslatenz. Trotzdem wird empfohlen QPC oder die Windows-Systemzeit als ETW-Zeitgeber zu verwenden, da die direkte Nutzung des CPU-Zykluszählers nicht unter allen Umständen sicher ist. Bevor dieses Verfahren für ETW genutzt wird, muss manuell sichergestellt werden, dass die Hard- und Software-Plattform einen sicheren Einsatz des CPU-Zykluszählers ermöglichen. Sind die TSC-Register der Rechenkerne nicht synchronisiert, können die in den Event-Objekten gespeicherten Auftrittszeitpunkte von Ereignissen, die auf verschiedenen CPU-Kernen ausgelöst wurden, nicht miteinander verglichen und nicht zeitlich geordnet werden. Ist die Betriebsfrequenz des TSC abhängig von der Taktfrequenz des Rechenkerns, kann diese über die Aufzeichnungszeit variieren, da die Taktfrequenz von einzelnen Prozessor-Rechenkernen im Betrieb lastbedingt verändert wird. Zudem besitzen dadurch die TSCs verschiedener CPU-Kerne eventuell unterschiedliche Betriebsfrequenzen. Dies alles führt wiederum zu Inkonsistenzen bezüglich der Vergleichbarkeit von Zeitangaben in den ETW-Events [Mic5b].

Um die Auflösung zu bestimmen, schreibt das ETW-System die Betriebsfrequenz des Zykluszählers in das Feld `CpuSpeedInMHz` der `TRACE_LOGFILE_HEADER`-Struktur, welcher die ETW-Session spezifiziert. Diese Information dient einem möglichen ETW-Consumer dazu, den Wert des Zeitstempels des Event-Objektes korrekt zu interpretieren [Mic5b][Mic5a].

CPU-Zykluszähler

basiert auf:	TSC
Typ:	Hardware-Zähler
Realzeit:	nein
Format:	64 Bit Zeitstempel
Auflösung Zählvariable:	1 Zyklus
zeitliche Auflösung:	hardwareabhängig ($<$ 0,3 ns möglich)
zeitliche Genauigkeit:	ca. 20 ns - 30 ns (Abhängig vom Pipeline-Zustand der CPU)
Nullpunkt:	nicht sicher
Stabilität:	nicht sicher (TSC kann geändert werden)

Tabelle 3.4: Steckbrief: CPU-Zykluszähler

3.5.3 Abstraktionsebene

Für ETW-Events kann nicht allgemein angegeben werden, auf welcher Abstraktionsebene die Ereignisse angesiedelt sind. Grund dafür ist, dass ETW-Ereignisse allgemein jegliche Vorgänge, jeglicher Abstraktionsebene beschreiben können. Stattdessen kann die Abstraktionsebene für konkrete Ereignistypen oder auch für ETW-Provider-Klassen angeben werden. Es existieren ETW-Provider, die Ereignisse der Systemebene wie Interrupts, Systemcalls, Kontextwechsel usw. melden. Wiederum andere Ereignisquellen erzeugen Event-Objekte für die unterschiedlichsten Arten des Thread-/Prozessverhaltens. Dabei existieren Provider und Ereignistypen, die feingranulare Einzelanfragen des Threads oder Prozesses an den Betriebssystem-Kernel melden, sowie auch Provider, die eher gröbere Aktionen im Thread-/Prozessverhalten signalisieren. Die Abstraktionsebene hängt somit sehr stark von den verwendeten Providern bzw. Provider-Klassen ab und deren Ereignistypen.

Einschränkend kann gesagt werden, dass sehr hochfrequente Ereignisse auf Ausführungsebene, wie Sprünge, Funktionsaufrufe oder die Verwendung bestimmter Maschinenbefehle normalerweise nicht durch ETW protokolliert werden können. Die Ereigniserzeugung mittels ETW besitzt einen gewissen Overhead, der zu groß sein könnte, um so extrem feingranulare Ereignisse zu erfassen.

Auf der anderen Seite der Abstraktionsebenen liegen die höher aggregierten Ereignisse, die eher semantische Vorgänge und den Zweck bestimmten Systemverhaltens versuchen zu beschreiben, als die einzelnen Aktionen des Verhaltens selbst. In ETW sind keine Provider-Klassen bekannt, die Ereignistypen dieser Abstraktionsebene erzeugen. Dies ist nicht verwunderlich, da ETW ursprünglich ein Werkzeug für die Ablaufverfolgung (Tracing) ist. Ereignisse auf hohen Meta-Ebenen fallen normalerweise nicht in diese Kategorie.

3.6 Architektur

Die Kern-Komponente des ETW-Systems ist Teil des Windows-Kernels. Sie verwaltet die aus Ereignis-Puffern bestehenden ETW-Sessions. In den ebenfalls im Kernel lokalisierten ETW-Sessions können ETW-Event-Objekte zwischengespeichert werden. Eine Session besteht jedoch nicht nur aus ihren Ereignis-Puffern sondern weiteren Kontrollstrukturen. So besitzt eine Session eine Konfiguration, die unter anderem spezifiziert, welche Ereignisse aus welchen Ereignis-Quellen in ihr gespeichert werden sollen.

Die Ereignisquellen werden als ETW-Provider bezeichnet. Die Rolle des ETW-Providers kann allgemein betrachtet jegliche Software des Systems einnehmen, die ETW-Ereignis-Objekte erzeugt und der ETW-Kern-Komponente im Kernel zur Verfügung stellt. ETW-Event-Objekte besitzen einen Ereignistyp, der in einer Provider-Klasse spezifiziert ist. Die eingehenden Ereignis-Objekte werden in die Puffer der ETW-Sessions kopiert, welche die Ereignisse der entsprechenden Provider-Klasse abonniert haben. Generell kann jeglicher Anwendungs-Code von Applikationen und Diensten, der im User-Mode ausgeführt wird, mittels ETW-Bibliotheken Ereignisse dem ETW-System bereitstellen und so die Rolle des ETW-Providers einnehmen. Somit kann auch im User-Mode ausgeführter Bibliotheks-Code von dieser Möglichkeit Gebrauch machen. Darüber hinaus kann Kernel-Code sowohl des Windows-Kernels selbst (`ntoskrnl.exe`, `win32k.sys`, `hal.dll`) als auch der Code geladener Kernel-Treiber über die ETW-Kernel-API Ereignisse erzeugen und dem ETW-System bereitstellen und somit zum ETW-Provider werden [AIRS21, S. 499ff][Mic6a][Yos19, min. 4:46][Gol17, min. 6:00].

Berechtigte Applikationen haben die Möglichkeit, über eine spezielle Controller-API das ETW-System zu steuern. Dies beinhaltet unter anderem das Anlegen und Konfigurieren neuer ETW-Sessions sowie das Starten und Stoppen von Ereignis-Aufzeichnungen. Solche ETW-Steuerungssoftware füllt die Rolle des ETW-Controllers aus [Yos19, min. 4:46][Gol17, min. 6:00].

Die in einer ETW-Session gesammelten Ereignis-Objekte können automatisch durch den Windows-Kernel in spezielle ETL-Dateien (Event Trace Logging) auf einem Massenspeicher geschrieben werden [AIRS21, S. 511ff].

Diese ETL-Dateien können von berechtigten Applikationen gelesen und die enthaltene Event-Folge verarbeitet werden. Solche Applikationen werden als ETW-Consumer bezeichnet. ETW-Consumer-Apps können alternativ Ereignisse auch direkt als Live-Ereignisstrom vom ETW-System im Kernel beziehen. Über eine spezielle Consumer-API können diese Apps benachrichtigt werden, wenn in den entsprechenden Sessions neue Ereignis-Objekte eingegangen sind. Über die API können die zwischengespeicherten Ereignis-Objekte abgerufen werden [MicAa][AIRS21, S. 512ff].

Die Latenz zwischen dem Erzeugen eines Ereignisses bei der Provider-Komponente und dem Erhalt auf Consumer-Seite kann bei einem Live-Ereignisstrom bis zu drei Sekunden betragen und geschieht in jedem Fall asynchron [Yos19, min. 26:00].

Möchte eine Software die selbst erstellten Event-Objekte selbst erhalten und verarbeiten, um über diese Methode ein eigenes Prozess-internes Nachrichtensystem umzusetzen, steht dafür eine besondere Variante von ETW bereit. Diese Variante sieht vor, dass ein Programm zugleich alle drei Rollen Provider, Controller und Consumer übernimmt und für den internen Austausch der Event-Objekte eine private ETW-Session benutzt. Die privaten ETW-Sessions sind nicht Teil des ETW-Systems im Kernel. Stattdessen werden sie von den Bibliotheken hinter der ETW-API innerhalb des User-Mode-Prozesses umgesetzt [Mic6a]. Auf diese spezielle Variante und die privaten ETW-Sessions wird in dieser Arbeit nicht weiter eingegangen, da sie für die Verhaltensüberwachung und Angriffserkennung nicht relevant sind.

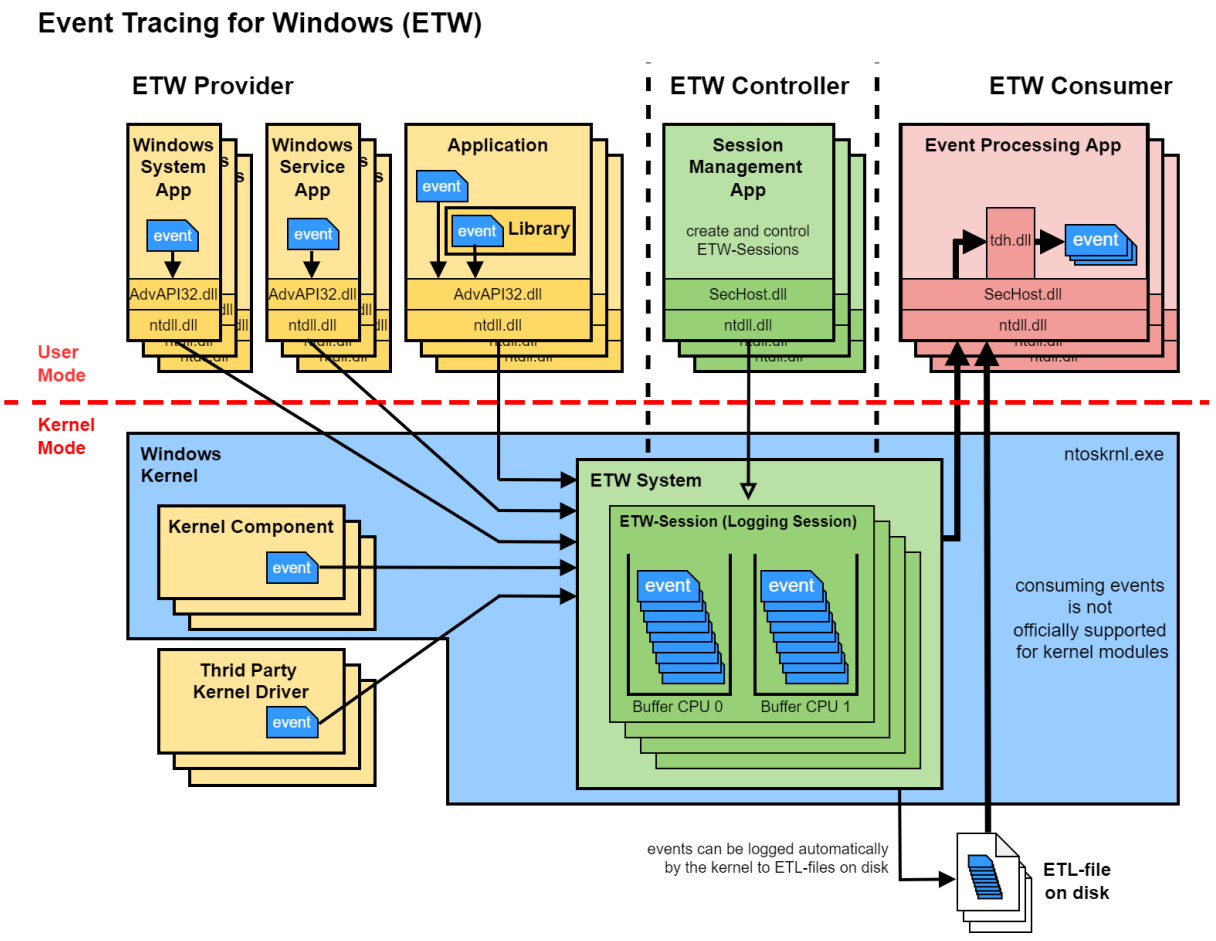

Abbildung 3.1: Event Tracing for Windows (ETW) - Architektur

3.6.1 Rolle: ETW-Provider

Komponenten, die ETW-Ereignisse erzeugen, nehmen im ETW-System die Rolle des ETW-Providers ein. Dies können Anwendungsprogramme, Bibliotheken oder auch Kernel-Komponenten wie Treiber oder zentrale Kernel-Funktionen wie Sheduler, Speicher-Manager, Objekt-Manager usw. sein. ETW-Provider-Komponenten können darin unterschieden werden, ob sie im User-Mode (Anwendungscode, Bibliothekscode) oder im Kernel-Mode (Kernel-Komponenten, Kernel-Treiber) ausgeführt werden.

Obwohl jedes der genannten Komponenten im Kontext eines Threads und Prozesses ausgeführt wird, wird jedoch meist nicht der Prozess als ETW-Provider bezeichnet, sondern die statische Code-Komponente, welche die Funktionalität, ETW-Ereignisse zu erzeugen, enthält. Dies erscheint logisch, da dieselbe Code-Komponente in mehreren verschiedenen Prozessen geladen sein und somit im Kontext verschiedener Prozesse und Threads zur Ausführung kommen kann. Die Kernel-Komponenten, welche die Rolle eines ETW-Providers übernehmen, sind in der Regel sogar Teil jedes Prozesses und können theoretisch in jedem Prozess- und Thread-Kontext ausgeführt werden. Nichtsdestotrotz wird jedes einzelne ETW-Ereignis in einem konkreten Thread- und Prozess-Kontext erzeugt und kann diesem zugeordnet werden [Mic8a].

ETW-Provider basieren auf einer Provider-Klasse, für die sie sich beim ETW-System anmelden. Die Provider-Klassen sind auf dem System installiert und spezifizieren jeweils eine Menge an Ereignistypen und deren Attribute. Diese definieren die Event-Objekt-Struktur für die Event-Objekte, die der Provider erstellt. Darüber hinaus spezifiziert eine Provider-Klasse weitere allgemeine Schlüsselwörter, die der Provider für jedes seiner erzeugten Ereignisse individuell spezifizieren kann [AIRS21, S. 506ff].

Ein ETW-Provider erzeugt meistens nur Event-Objekte, wenn diese auch angefordert werden. Die Provider-Klasse mit der die Provider-Komponente sich beim ETW-System angemeldet hat, kann in mehreren ETW-Sessions abonniert sein. Das Abonnement einer Provider-Klasse durch eine ETW-Session bedeutet, dass die Session Event-Objekte der Ereignistypen, die in der Provider-Klasse spezifiziert sind, protokollieren möchte. Nur wenn mindestens eine ETW-Session die ETW-Provider-Klasse abonniert hat, werden der Provider, die sich für die Klasse angemeldet haben, aktiviert. Ein ETW-Provider sendet normalerweise nur im aktivierten Zustand Event-Objekte an das ETW-System [AIRS21, S. 508ff][Mic8a]. Das Abonnement kann noch spezifischer Konfiguriert werden. So kann eine Session spezifizieren, welche Gruppen von Ereignistypen, sie beziehen möchte. Die Gruppen werden durch Bitmasken mit Schlüsselwörtern, welche die Provider-Klasse definiert, konfiguriert. Je nach Provider-Klassen-Typ ist es möglich, dass auch die aggregierten Bitmasken bei der Provider-Aktivierung den Providern mitgeteilt werden, sodass

diese unter Umständen nicht alle Ereignistypen, die in der Klasse spezifiziert sind, erzeugen, sondern stattdessen nur die benötigten Event-Objekte, die auch in mindestens einer ETW-Session gesammelt werden [Mic8b].

3.6.2 Rolle: ETW-Controller

Software-Komponenten können die Rolle eines ETW-Controllers einnehmen. In der Regel handelt es sich dabei um Anwendungen. Jedoch wären auch Bibliotheken oder Kernel-Treiber denkbar, die als ETW-Controller fungieren. Als ETW-Controller wird dabei jede Software bezeichnet, die ETW-Sessions anlegt, verwaltet, konfiguriert und steuert [Mic6a]. Oft implementiert ETW-Analysesoftware, mit der ETW-Protokollierungen aufgezeichnet und im Anschluss ausgewertet werden können, sowohl die Rolle des ETW-Consumers als auch die Rolle des ETW-Controllers zugleich. Unter Windows existieren auch Kommandozeilen-Anwendungen, die zum Anlegen und Steuern von ETW-Sessions benutzt werden können und somit die Rolle des ETW-Controllers einnehmen. Beispiele sind `logman` [Mic2a] und `xperf` [Mic2b]. Auch die `PowerShell` stellt Befehle bereit, um ETW-Sessions zu konfigurieren und zu steuern [Mic2c].

Der klassische ETW-Controller ist ein User-Mode-Prozess, der über die ETW-Controller-API mit dem ETW-System im Kernel kommuniziert. Er legt ETW-Sessions an und konfiguriert diese. Dazu gehört die Auswahl der ETW-Ereignistypen, die durch die Session protokolliert werden sollen. Diese werden zum einen durch das Abonnieren von ETW-Provider-Klassen für eine Session definiert. Eine Provider-Klasse beschreibt eine Menge an ETW-Ereignistypen die semantisch zusammengehören und in der Regel aus der gleichen Quelle stammen. Zudem anderen kann der Controller für jede abonnierte Provider-Klasse in der Session Filter-Regeln spezifizieren, die definiere, welche eingehenden Ereignisse in der Session gesammelt werden sollen [Mic6a].

Der ETW-Controller hat die Aufgabe, die erstellten Sessions zu aktivieren und zu deaktivieren bzw. die Protokollierung zu starten und zu stoppen.

Das Abonnieren von Provider-Klassen und Anpassen der Filterregeln kann im aktiven Logging-Betrieb stattfinden, also während eine ETW-Session aktiv gestartet ist. Das ETW-System wird dadurch unter Umständen die Provider der Klassen während des Protokollierungsbetriebs aktivieren, deaktivieren und ihre Filtereinstellungen aktualisieren [AIRS21, S. 508ff][Mic9a].

3.6.3 Rolle: ETW-Consumer

Die dritte Rolle, die die ETW-Architektur vorsieht, ist der ETW-Consumer. Software-Komponenten, die ETW-Ereignisse von ETW-Sessions live beziehen oder aufgezeichnete Protokollierungen aus ETL-Dateien abrufen, um diese weiterzuverarbeiten, nehmen die Rolle des ETW-Consumers ein. In der Regel sind dies User-Mode-Anwendungsprogramme. Die Schnittstelle zum Abrufen der Ereignisse steht offiziell nur in der User-Mode-API zur Verfügung, sodass für Kernel-Komponenten keine offiziell Möglichkeit vorgesehen ist, ETW-Event-Objekte zu beziehen und somit die Rolle des ETW-Consumers einzunehmen. Selbstverständlich können Software-Komponenten auch mehrere ETW-Rollen zugleich einnehmen. So ist ETW-Tracing-Software meistens sowohl Controller, um ETW-Sessions anzulegen und zu konfigurieren, als auch Consumer, um die Ereignisse zu empfangen und dem Benutzer darzustellen oder Informationen daraus zu extrahieren [Mic6a][AIRS21, S. 512ff].

Ein Consumer-Prozess kann die Events einer Session direkt beziehen, wenn die Session dies ermöglicht (Real-Time-Tracing-Session). In diesem Fall meldet er sich für den direkten Erhalt von Event-Objekten einer speziellen Session über die ETW-Consumer-API beim ETW-System im Kernel an. Er wird daraufhin automatisch vom ETW-System benachrichtigt, wenn neue Event-Objekte in der Session vorliegen. Diese kann der Consumer über dieselbe ETW-User-Mode-API auch beziehen. Zwischen der Erzeugung eines ETW-Events im Provider und der Benachrichtigung des Consumers, können bis zu drei Sekunden vergehen [MicAa][AIRS21, S. 512ff]. In den meisten Fällen ist die Latenz jedoch geringer. Dennoch sollte erwähnt werden, dass die Latenz unterschiedlich ausfallen kann und im ungünstigen Fall im Bereich von Sekunden liegen kann, da die Begriffe Real-Time-Tracing und Live-Event-Session beim Leser die Erwartung implizieren könnten, dass die Latenz sehr kurz und im Sinne der Echtzeit deterministisch sei. Beide Anforderungen erfüllt das asynchron arbeitende ETW-System nicht.

Alternativ kann das ETW-System im Kernel die Event-Objekte selbstständig in ETL-Dateien speichern. Die Dateien enthalten eine Folge von ETW-Event-Objekten ähnlich einer Protokolldatei. Eine Software, die ETW-Event-Objekten aus ETL-Dateien ausliest wird auch als ETW-Consumer bezeichnet. Dies hat auch damit zu tun, dass dafür offiziell die gleiche User-Mode-API verwendet wird, wie für das direkte Auslesen der ETW-Event-Objekte aus der Session [MicAa].

Unabhängig ob die Event-Objekte direkt aus der Session oder aus einer ETL-Datei bezogen werden, sind diese in einem offiziell nicht dokumentierten Format binär kodiert (serialisiert). Die ETW-User-Mode-API stellt dem Consumer Funktionen zur Verfügung, die Daten der Event-Objekte zu dekodieren, um die Ereignis-Attribute zu erhalten. Of-

fiziell ist dies der einzige Weg, die Daten aus Event-Objekten zu erhalten [AIRS21, S. 513ff].

3.6.4 ETW-Provider-Klassen

Die Provider-Klasse beschreibt eine Menge an ETW-Ereignistypen und deren Attribute. Hinzu kommen Kategorien und Schlüsselwörter, denen die Ereignistypen zugeordnet sind. Diese stellen die Basis für Filter-Regeln dar, welche Teil der Konfiguration einer ETW-Session sein können. Es gibt jedoch auch Kategorien und Schlüsselwörter, die nicht direkt einem Ereignistyp zugeordnet sind, sondern stattdessen vom Provider bei der Erzeugung eines Event-Objektes individuell für das Ereignis vergeben werden können. Da Filter-Regeln in der ETW-Sessions immer 32-Bit-breite Bitmasken sind, besteht eine wichtige Limitierung. Die Anzahl der in einer Provider-Klasse spezifizierten Schlüsselwörter und Kategorien darf nicht größer 32 sein [Mic6b].

In der Literatur wird für Provider-Klassen und die Rolle des ETW-Providers gleichermaßen der Begriff Provider benutzt, was die Unterscheidung beider Sachverhalte schwer macht. Aus diesem Grund wird für die bessere Unterscheidbarkeit in dieser Arbeit für die Spezifikation der ETW-Ereignistypen der Begriff Provider-Klasse anstatt der Begriff Provider verwendet.

Die Provider-Klasse beschreibt die Datenformate der Event-Objekte für die verschiedenen Ereignistypen als geordnete Attribut-Listen. Für jeden Ereignistyp der Provider-Klasse kann in der Klasse eine Attribut-Liste spezifiziert sein. Jedes Attribut besitzt einen Bezeichner, einen Datentyp und eine Position in der Liste. Die Position eines Attributs trägt eine entscheidende Rolle für die Event-Erzeugung. Der Provider muss die Werte für die Attribute in genau dieser Reihenfolge an die ETW-API übergeben, die daraus ein binäres ETW-Event-Objekt serialisiert, um dieses in der ETW-Session im Kernel speichern zu können. Als Datentypen für die Attribute stehen folgende Basisdatentypen zur Verfügung: `AnsiString`, `UnicodeString`, `Int8`, `UInt8`, `Int16`, `UInt16`, `Int32`, `UInt32`, `Int64`, `UInt64`, `Float`, `Double`, `Boolean`, `GUID`, `FILETIME`, `SYSTEMTIME`, `SID`, `Pointer`, `Binary` (Byte-Array frei definierbarer fester Länge; Länge muss in der Definition angegeben werden oder ein Verweis auf ein anderes Attribut sein, dass die Länge spezifiziert). Zusätzlich existiert ein Struktur-Typ, der es erlaubt, Attribute zu gruppieren. Zudem können von jedem Typ (inklusive dem Struktur-Typ) Listen-Attribute spezifiziert werden. Bei diesen muss die Länge der Liste in gleicher Weise wie die Länge der Byte-Arrays spezifiziert werden. Sie kann also fest oder variabel sein. Eine variable Länge bedeutet, dass sie sich erst zur Laufzeit ergibt. In diesem Fall spezifiziert wiederum ein anderes Attributs-Feld die Länge. Bei der Definition der Ereignistypen ist zu

beachten, dass die Größe der Gesamtdaten eines ETW-Events nur maximal 64 KiB betragen darf. Dies limitiert die Anzahl und verwendeten Datentypen der Attribute eines ETW-Event-Objektes. Versucht ein Provider über die ETW-API ein Event-Objekt zu erstellen, dass größer als 64 KiB ist, wird dieses vom ETW-System verworfen, da nur binäre Blobs von der maximalen Größe 64 KiB in einer ETW-Session gespeichert werden können [Mic7a].

Beispiel einer Attribut-Liste beschrieben in einem Instrumentierungs-Manifest:

Listing 3.1: Definition eines ETW-Eventtyps in einem Provider-Manifest (Quelle: Microsoft API-Dokomentation [Mic7a])

```
...
<events>
    <event value    = "1"
           level    = "win:Informational"
           keywords = "Remote Read"
           task     = "Connect"
           template = "event1_attributes_template"
           channel  = "c1"
           symbol   = "TRANSFER_SCHEDULE_EVENT"
           message  = "\$(string.Event.XferSchedule)"
    />
</events>
...
<template tid="event1_attributes_template">
    <data name="TransferName"   inType="win:UnicodeString" />
    <data name="ErrorCode"      inType="win:Int32" />
    <data name="FilesCount"     inType="win:UInt16" />
    <data name="Files"          inType="win:UnicodeString"  count="FilesCount" />
    <data name="BufferSize"     inType="win:UInt32" />
    <data name="Buffer"         inType="win:Binary"         length="BufferSize"/>
    <data name="Certificate"    inType="win:Binary"         length="11" />
    <data name="IsLocal"        inType="win:Boolean" />
    <data name="Path"           inType="win:UnicodeString" />
    <data name="ValuesCount"    inType="win:UInt16" />
    <struct name="Values"   count="ValuesCount" >
        <data name="Value"  inType="win:UInt16" />
        <data name="Name"   inType="win:UnicodeString" />
    </struct>
</template>
...
```

ETW-Provider-Klassen müssen im Windows-System installiert werden, damit sie Providern, Controllern und Consumern gleichermaßen systemweit zur Verfügung stehen. Installierte Provider-Klassen werden von ETW-Providern für das Kodieren der Ereignisdaten verwendet. ETW-Controller benötigen die Informationen der Provider-Klassen zum Konfigurieren der ETW-Sessions. ETW-Consumer können mittels der Informationen der Provider-Klassen die eingegangenen ETW-Ereignis-Objekte dekodieren und die enthaltenen Daten korrekt interpretieren.

Ein frisch installiertes Windows bringt eine große Menge von bereits installierten ETW-Provider-Klassen mit. Diese werden standardmäßig von einer großen Anzahl an Providern in den meisten Windows-Komponenten genutzt [Gol17, min. 8:27]. Sowohl der Windows-Kernel als auch zentrale User-Mode Applikationen und Services des Windows-Systems, sowie die Windows-Laufzeit-Bibliotheken und -Subsysteme signalisieren über diese Provider-Klassen Ereignisse [TKG21a, S. 5]. Weitere Dritt-Software wie beispielsweise Anwendungsprogramme oder Gerätetreiber können bei ihrer Installation weitere eigene Provider-Klassen im System registrieren, um diese für die Erzeugung eigener ETW-Events zu nutzen.

Die auf einem Windows-System installierten Provider-Klassen und deren Eigenschaften wie Ereignistypen, Schlüsselwörter usw. können über die Kommandozeile abgefragt werden (z.B. mittels `wevtutil`) [Mic2d]. Die Software ETW-Explorer von Pavel Yosifovich bietet eine grafische Oberfläche, um die registrierten Provider-Klassen aufzulisten und deren Inhalt anzuzeigen. Dabei werden die Ereignistypen und deren Attribute, Schlüsselwörter und Kategorien tabellarisch dargestellt [Yos21].

ETW-Provider-Klassen besitzen eine eindeutige GUID, über die sie adressiert werden können. Zusätzlich tragen sie einen für den Menschen lesbaren Bezeichner, der aber nicht für die Identifikation verwendet wird. Wie ein System-Objekt besitzt jede ETW-Provider-Klasse einen Security-Deskriptor. Dieser enthält die klassischen Security-Attribute einer Systemressource [AIRS21, S. 506ff]. Dies sind beispielsweise ein Besitzer-Attribut und eine Access-Control-List (ACL). Letztere regelt die Zugriffsberechtigungen bezüglich der ETW-Provider-Klasse. Die Security-Deskriptoren aller installierter ETW-Provider-Klassen sind unter dem Registry-Schlüssel `HKLM\System\CurrentControlSet\Control\WMI\Security\` in der Windows-Registrierungsdatenbank als einzelner Wert pro Deskriptor gespeichert. Der gesamte Deskriptor ist als binäre Bytefolge serialisiert abgelegt [AIRS21, S. 522ff].

Spezielle Berechtigungen einer ETW-Provider-Klasse, die für Benutzerkonten gewährt und entzogen werden können:

`TRACELOG_GUID_ENABLED`	Erlaubt einem Controller diese ETW-Provider-Klasse für die Session zu abonnieren (die Provider-Klasse zu aktivieren)
`TRACELOG_REGISTER_GUIDS`	Erlaubt einem Provider sich für diese ETW-Provider-Klasse anzumelden (sich für die Provider-Klassen-GUID zu registrieren)

Tabelle 3.5: spezielle ACL-Berechtigungen einer ETW-Provider-Klasse [AIRS21, S. 522ff]

Es existieren vier verschiedene Typen von ETW-Provider-Klassen, die sich in ihrer internen Implementierung und dem Registrierungsverfahren bzw. der Installationsmethode

unterscheiden. Sie werden als „Manifest-Provider-Klassen“, „MOF-Provider-Klassen“, „Trace-Logging-Provider-Klassen“ und „WPP-Provider-Klassen“ bezeichnet [AIRS21, S. 499ff][Mic6a].

3.6.4.1 Manifest-Provider-Klassen

Eine auf einem Manifest basierende Provider-Klasse wird mittels einer Manifest-Datei im XML-Format spezifiziert. Das Manifest enthält die gesamte Beschreibung der Provider-Klasse inklusive aller enthaltenen Ereignistypen, Kategorien, Schlüsselwörter usw., die durch die Provider-Klasse definiert werden. Bei der Installation der Provider-Klasse wird diese als Satz von Registry-Schlüsseln in der Registrierungsdatenbank abgelegt. Die in dieser Weise registrierte Provider-Klasse wird von Provider- und Consumer-Komponenten zur Kodierung und Dekodierung der Ereignisdaten verwendet. Provider, Controller und Consumer beziehen die benötigten Informationen zur Provider-Klasse aus der Registry. Alle registrierten Manifest-Provider-Klassen finden sich in der Windows-Registry unter folgendem Schlüssel:
`HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\WINEVT\Publishers\`
[Mic8b][AIRS21, S. 499ff].

Manifest-Provider-Klassen können maximal von acht ETW-Sessions gleichzeitig abonniert sein [Mic6b].

3.6.4.2 MOF-Provider-Klassen

MOF-Provider-Klassen gehören zum Altbestand (legacy) von ETW und basieren auf dem WMI-System (Windows Management Instrumentation) wie auch einige Teile des ETW-Kern-Systems. MOF-Provider-Klassen nutzen MOF-Klassen (Managed Object Format) für die Spezifikation. Die MOF-Klassen beschreiben die Ereignistypen und Schlüsselwörter der Provider-Klasse. Das Managed-Object-Format (MOF) ist eine Technologie des WMI-Systems. Die Informationen der MOF-Klassen werden zum Kodieren der Ereignisdaten im Provider und zum Dekodieren der Daten im Consumer benutzt. Bei der Installation einer MOF-Provider-Klasse werden die MOF-Klassen im WMI-Repository im Windows-Kernel abgelegt. Aus diesem beziehen Provider, Controller und Consumer die benötigten Informationen [Mic8c][AIRS21, S. 499ff].

MOF-Provider-Klassen sind der Vorgänger der Manifest-basierten Provider-Klassen und werden durch diese abgelöst. Trotz alledem existieren eine Reihe von MOF-Provider-Klassen, die für eine Systemüberwachung sehr relevant sind. Es ist aber davon auszugehen, dass in zukünftige Windows Versionen keine neuen MOF-Provider-Klassen mehr hinzukommen und stattdessen Manifest-Provider-Klassen hinzugefügt werden.

MOF-Provider-Klassen konnten ursprünglich nur von einer ETW-Session gleichzeitig abonniert sein. Mit Windows 8 wurde diese Limitierung aufgehoben. Seit dem kann eine MOF-Provider-Klasse genauso wie eine Manifest-Provider-Klasse von acht ETW-Sessions gleichzeitig abonniert sein [Mic6b].

3.6.4.3 Trace-Provider-Klassen

Trace-Provider-Klassen müssen nicht im System installiert werden. Die Daten eines Event-Objektes eines bestimmten Ereignistyps einer Trace-Provider-Klasse werden in einem selbstbeschreibenden Format gespeichert, welches in den Daten des Event-Objektes selbst alle zusätzlichen Informationen enthält, die notwendig sind, damit die Consumer-Komponente die Daten des Event-Objektes korrekt dekodieren und interpretieren kann. Trace-Provider-Klassen werden als Profiling-Technologie bei der Software-Entwicklung eingesetzt. Daher müssen die Klassen in der Regel auch nicht systemweit bekannt sein, sondern nur innerhalb des Entwicklungsprojektes in der Entwicklungsumgebung (Visual Studio). Diese fungiert gleichermaßen als Controller und Consumer, während die zu entwickelnde Anwendung die Rolle des Providers einnimmt [AIRS21, S. 499ff].

3.6.4.4 WPP-Provider-Klassen

WPP-Provider-Klassen basieren auf der Technologie Windows-Software-Trace-Pre-Processor (WPP). WPP ist eine Profiling-Technologie, die bei der Software-Entwicklung eingesetzt werden kann. Daher müssen WPP-Provider-Klassen ebenfalls nicht systemweit installiert werden. Eine WPP-Provider-Klassen wird durch eine TMF-Datei (trace message format) spezifiziert, welche die Ereignistypen und deren Attribute festlegt. Die TMF-Spezifikation wird zum Kodieren der Event-Objekt-Daten beim Provider verwendet und dient gleichermaßen möglichen Consumer-Komponenten zum Dekodieren der Daten der Event-Objekte [AIRS21, S. 499ff].

3.6.5 ETW-Sessions

ETW-Sessions sind die zentralen Objekte, die das ETW-Kern-System im Windows-Kernel verwaltet. Sie beinhalten einen Pool von mehreren Ereignis-Puffern, in denen ETW-Ereignisse gespeichert werden. Für jeden logischen Prozessor des Systems enthält jede ETW-Session einen separaten Ereignis-Puffer. Somit können Events von verschiedenen Rechenkernen aus gleichzeitig ins ETW-System eingetragen werden, ohne das Blockierungen (Locks) notwendig sind [AIRS21, S. 502ff]. ETW-Sessions sind zudem die Host-Strukturen der ETW-Logger-Threads. Für jede aktive ETW-Session existiert ein Logger-Thread im System-Prozess. Der Logger-Thread besitzt die Aufgabe, den Inhalt

der Puffer seiner Session in die ETL-Datei zu schreiben oder die Consumer-Applikationen zu benachrichtigen und diesen die Event-Daten zuzustellen [AIRS21, S. 511ff].

ETW-Sessions werden von Controller-Apps zum Sammeln von ETW-Ereignissen für Consumer-Apps erstellt und verwaltet. Der Controller konfiguriert die Session und kann das Loggen von Ereignissen starten und stoppen. Eine Session kann somit aktiv oder nicht aktiv sein, je nachdem ob diese gestartet ist und Events sammelt oder nicht [Mic9a][Mic6b].

Jede ETW-Session besitzt einen eindeutigen Namen, über den sie adressiert werden kann [Mic9a]. Intern besitzt jede ETW-Session zusätzlich eine GUID, die über die Konfigurations-Datenstrukturen abgefragt und bei der Erstellung optional festgelegt werden kann. Die Controller-API verwendet trotzdem primär den Bezeichner der Session als Parameter, um diese zu identifizieren [Mic5a][Mic9c].

Eine ETW-Session besitzt neben ihrem Namen und der GUID noch viele weitere wichtige Eigenschaften, die durch eine Controller-App spezifiziert werden können. Je nach Eigenschaft kann dies beim Erstellen der Session oder auch während des aktiven Logging-Betriebs geschehen [Mic9a].

Beispielsweise kann für jede ETW-Session konfiguriert werden, ob diese als Real-Time-Trace-Session, als Buffered-Trace-Session oder beides betrieben werden soll. Bei einer Real-Time-Trace-Session werden die eingehenden Ereignisse nur im Puffer zwischengespeichert, damit sie von Consumer-Applikationen entnommen werden können. Verbundene Consumer-Applikationen werden benachrichtigt, wenn neue Ereignisse eingegangen sind und müssen sich diese daraufhin selbstständig abholen. Passiert dies nicht ausreichend schnell, kann es beim Volllaufen der Puffer dazu kommen, dass ältere Ereignisse verloren gehen. Beim Betrieb einer Buffered-Trace-Session wird der Pufferinhalt bei Bedarf automatisch durch den Logger-Thread der Session in eine ETL-Datei geschrieben. Buffered-Trace-Sessions können jedoch zusätzlich auch einen Live-Ereignisstrom wie die Real-Time-Trace-Session liefern. Zudem kann unter anderem die Puffergröße einer ETW-Session und die maximale Größe der ETL-Datei konfiguriert werden. [AIRS21, S. 502ff][Mic6b][Mic9a].

Die wichtigste Eigenschaft stellt jedoch der Satz an abonnierten Provider-Klassen mit zusätzlichen Filter-Konfigurationen dar. Sie definieren, welche ETW-Ereignistypen in der Session gesammelt werden. Die Konfiguration bezüglich abonnierter Provider-Klassen und der Ereignistyp-Filter kann während einer laufenden Protokollierung durch den Controller angepasst werden. Die Filterregeln einer ETW-Session können für jede abonnierte Provider-Klasse separat festgelegt werden. Dies geschieht mittels der Schlüsselwörter,

welche die Provider-Klasse definiert und die den ETW-Events zugeordnet sind, über 32-Bit-breite Bitmasken. Die Bedeutung der einzelnen Bits ist Provider-Klassen-spezifisch und wird durch die Klasse mittels einer Bit-Schlüsselwort-Zuordnung definiert [Mic9a].

Für folgende allgemeinen Attribute können Filter konfiguriert werden:

`Level`	Wert	Das Ereignis muss einen gleichen oder kleineren Level-Wert besitzen (Verbose: 5, Info: 4, Warning: 3, Error: 2, Critical: 1)
`MatchAllKeyword`	Bitmaske	Das Ereignis muss allen Keywords, die durch die Bitmaske spezifiziert sind, zugeordnet sein.
`MatchAnyKeyword`	Bitmaske	Das Ereignis muss mindestens einem der Keywords, die durch die Bitmaske spezifiziert sind, zugeordnet sein.

Tabelle 3.6: Filtereinstellungen für abonnierte ETW-Provider-Klassen [Mic9a]

Eine weitere Eigenschaft jeder ETW-Session ist der Security-Deskriptor. So besitzt eine ETW-Session ein Besitzer-Attribut, welches auf das Benutzerkonto verweist, unter welchem die ETW-Session angelegt wurde. Wurde der Besitzer nicht geändert, entspricht dies dem Benutzerkonto, in dessen Kontext der Controller-Prozess ausgeführt wurde, der die Session erstellt hat. Zusätzlich gehört zum Security-Deskriptor eine Access-Control-List (ACL), welche die Berechtigungen für die verschiedenen Benutzerkonten hinsichtlich des Zugriffs auf die Session regelt. Die Security-Deskriptoren aller erstellter ETW-Sessions sind unter dem Schlüssel `HKLM\System\CurrentControlSet\Control\WMI\Security\` in der Registry als einzelner Wert pro Deskriptor gespeichert. Der gesamte Deskriptor ist als binäre Bytefolge serialisiert im Werte-Feld abgelegt [AIRS21, S. 522ff].

Spezielle ACL-Berechtigungen einer ETW-Session, die für Benutzerkonten gewährt und entzogen werden können:

`WMIGUID_QUERY`	Erlaubt einem Prozess Informationen über die Session abzufragen
`WMIGUID_NOTIFICATION`	Erlaubt einem Controller weitere Provider-Klassen der Session hinzuzufügen (senden von Provider-Aktivierungs-Benachrichtigungen)
`TRACELOG_CREATE_REALTIME`	Erlaubt einem Controller die Session als Real-Time-Trace-Session zu starten
`TRACELOG_CREATE_ONDISK`	Erlaubt einem Controller die Session als Buffered-Trace-Session zu starten
`TRACELOG_LOG_EVENT`	Erlaubt einem Provider Event-Objekte in diese Session einzutragen (nur relevant, wenn die Session im Secure-Mode ausgeführt wird)
`TRACELOG_ACCESS_REALTIME`	Erlaubt einem Consumer Event-Objekte aus dieser Session zu lesen (Real-Time-Trace-Session)

Tabelle 3.7: spezielle ACL-Berechtigungen einer ETW-Session [AIRS21, S. 522ff]

Die Anzahl an gleichzeitig parallel existierenden ETW-Sessions ist begrenzt. Aktuelle Windows Systeme (>= Vista) erlauben offiziell 64 parallele ETW-Sessions. Nicht offiziell dokumentiert ist, dass die maximale Anzahl parallel existierender ETW-Sessions angepasst werden kann. Windows-Internals beschreibt, dass der Grenzwert über den Registry-Wert `HKLM\System\CurrentControlSet\Control\WMI\EtwMaxLoggers` geändert werden kann. Laut Windows-Internals kann dieser Wert zwischen 32 und 256 variiert werden [AIRS21, S. 502ff].

3.6.6 ETW-System-Logging

Neben den normalen ETW-Provider-Klassen und normalen ETW-Sessions, die in den vorangegangenen Abschnitten beschrieben wurden, existieren auf Windows-Systemen noch spezielle System-Provider-Klassen und System-Sessions. System-Provider-Klassen werden nicht im System installiert und sind auch nicht in der Registry oder im WMI-Repository registriert. Stattdessen handelt es sich um eine feste Menge an, im ETW-System konstant definierter, Provider-Klassen. In aktuellen Windows-Versionen existieren genau zwei dieser System-Provider-Klassen. Dies ist die „NT Kernel Trace"-Provider-Klasse und die „Circular Kernel Context"-Provider-Klasse. Die System-Provider-Klasse „NT Kernel Trace" wird seit Windows 10 auch als „SystemTraceProvider" bezeichnet [AIRS21, S. 516ff][MicBa][MicBb]. Für System-Provider-Klassen können sich keine Provider-Komponenten von Drittanbieter-Software anmelden. Die beiden System-Provider-Klassen stehen ausschließlich Provider-Funktionen im Kernelcode zur Verfügung. Wie der Mechanismus der Nachrichtenerzeugung dieser System-Provider funktioniert, ist nicht durch Microsoft beschrieben. Ihre Ereignistypen sind jedoch zum Teil offiziell dokumentiert, sodass diese zwei System-Provider-Klassen als Event-Quellen genutzt werden können. Aus der Dokumentation der zwei System-Provider-Klassen geht hervor, dass die Beschreibungen der Event-Objekt-Formate für die verschiedenen Ereignistypen auf MOF-Klassen basieren. Die zwei System-Provider-Klassen sind also eine spezielle Form von MOF-Provider-Klassen, die im System fest verankert sind [Mic9b][Mic9c].

System-Provider-Klassen

Name	GUID
NT Kernel Trace (SystemTraceProvider)	9E814AAD-3204-11D2-9A82-006008A86939
Circular Kernel Context	54DEA73A-ED1F-42A4-AF71-3E63D056F174

Tabelle 3.8: System-Provider-Klassen

System-Provider-Klassen können nicht von normalen ETW-Sessions abonniert werden. Stattdessen sieht ETW neben den normalen ETW-Sessions zusätzlich noch spezielle ETW-System-Sessions vor. Seit Windows 8 können System-Sessions im Prinzip durch

eine Controller-Anwendung wie reguläre ETW-Sessions angelegt, verwaltet, gestartet und gestoppt werden, jedoch bestehen für System-Sessions spezielle Limitierungen. So kann eine System-Session nur System-Provider-Klassen abonnieren. Zudem darf immer nur eine System-Provider-Klasse gleichzeitig abonniert sein. Diese Limitierung wiegt nicht schwer, da insgesamt mit der „NT Kernel Trace"-Provider-Klasse und der „Circular Kernel Context"-Provider-Klasse nur zwei System-Provider-Klassen zur Verfügung stehen. Zudem können systemweit nur maximal acht System-Sessions durch ETW-Controller erzeugt werden [MicBa][AIRS21, S. 516ff].

Von den acht verfügbaren System-Sessions sind bereits zwei Plätze fest vergeben, sodass nur noch sechs System-Sessions durch Controller-Anwendungen angelegt werden können. Die zwei dauerhaft existierenden ETW-System-Session sind die „NT Kernel Logger"-Session, welche die Events der „NT Kernel Trace"-Provider-Klasse bezieht und die „Circular Kernel Context Logger"-Session, welche die Events der „Circular Kernel Context"-Provider-Klasse sammelt. Beide System-Sessions sind fest konfiguriert und dauerhaft im ETW-System angelegt. Sie können nicht entfernt werden und ihre Konfiguration bezüglich abonnierter Provider-Klassen kann nicht verändert werden. Es ist jedoch einer Controller-Anwendung möglich, die zwei vorhandenen System-Sessions zu starten, falls diese nicht bereits aktiv sind [AIRS21, S. 516ff][TKG21a, S. 25][TKG21b, min. 21:18]. Zudem kann die Keyword-Filter-Bitmaske für die eine abonnierte System-Provider-Klasse durch einen Controller angepasst werden. Auch Consumer-Anwendungen können sich als ETW-Consumer für die zwei vorhandenen System-Sessions anmelden und Event-Objekte aus diesen beziehen. Sollten die zwei vorhandenen System-Sessions noch nicht von anderen Anwendungen benutzt werden, können diese alternativ auch zum System-Logging verwendet werden, anstatt neue System-Sessions anzulegen. Vor Windows 8 war es einer Controller-Anwendung nicht möglich, eigene System-Sessions anzulegen. Zu dem Zeitpunkt existierten nur die zwei vorhandenen System-Sessions, die für das System-Logging zwangsläufig mitbenutzt werden mussten. Da die Verwendung derselben Session durch mehreren Controller- und Consumer-Instanzen nicht zuverlässig funktioniert, wird ab Windows 8 empfohlen, eigene System-Sessions unter anderem Namen anzulegen, anstatt die vorhandenen Sessions zu nutzen. Dadurch kann älterer Software aus dem Weg gegangen werden, welche die vorhandenen System-Sessions für das System-Logging verwendet [Mic9c][MicBb].

Die zwei System-Provider-Klassen „NT Kernel Trace" und „Circular Kernel Context" liefern zwar ausschließlich Ereignisse aus dem Windows-Kernel, jedoch basieren Kernel-Provider nicht zwangsläufig auf diesen zwei Klassen. Auf einem installierten Windows-System existiert eine große Anzahl an normalen Provider-Klassen, die Kernel-Ereignisse spezifizieren. Darunter sind sowohl Ereignisse von verschiedenen Treibern als auch von den Kernel-Basis-Funktionen des Windows-Kernel selbst. Darüber hinaus stehen mit dem Windows 10 SDK Build 20348 alle Ereignisse der System-Provider-Klasse „NT Kernel

Trace“ auch als mehrere normale Manifest-basierte Provider-Klassen zur Verfügung, die von normalen ETW-Sessions abonniert werden können. Somit liegt die Vermutung nahe, dass das System-Logging-Konzept mit System-Provider-Klassen und System-Sessions, welches auf MOF basiert, zukünftig in neueren Windows-Versionen zum Altbestand (Legacy) wird und durch die Verwendung der neueren Manifest-basierten Provider-Klassen vollständig ersetzt werden kann [MicBa].

3.6.7 ETW-Event-Objekte

Jedes ETW-Event-Objekt besteht aus einem allgemeinen ETW-Event-Header und den Event-Daten, die abhängig vom Ereignistyp sind. Zusätzlich können in einem Event-Objekt optional noch weitere allgemeine Informationen gespeichert werden (Stacktrace, SID usw.). Die Größe der Event-Daten ist Abhängig von der Anzahl und den Datentypen der Attribute des Ereignistyps, den das Event-Objekt angehört [Mic5c]. Ein ETW-Ereignis darf jedoch insgesamt mit Header und Daten maximal 64 KiB groß sein. Erzeugt ein ETW-Provider größere Event-Objekte, werden diese vom ETW-System verworfen [Mic8b]. Dies ist begründet durch die WMI-Pufferstruktur, in der die ETW-Objekte innerhalb der ETW-Session abgelegt werden. Diese erlaubt nur die Speicherung von Daten-Blobs mit einer Maximallänge von 64 KiB [AIRS21, S. 509ff]. Der Header ist bei jedem ETW-Event-Objekt nach dem gleichen Schema aufgebaut.

ETW-Event-Header

USHORT	Size	Größe des Event-Records
USHORT	HeaderType	Reserved (undokumentiert)
USHORT	Flags	Spezielle Eigenschaften des Ereignisses Bit 0: extendet info (Event-Record besitzt ExtendetInfo-Daten) Bit 1: private session (privates Ereignis) Bit 2: string only (Event-Daten sind eine Zeichenkette) Bit 3: trace message (Event ist WPP-Event) Bit 4: no cpu time (siehe Zeitfelder unten) Bit 5: 32 bit header (32Bit-Prozess erzeugte das Event manuell) Bit 6: 64 bit header (64Bit-Prozess erzeugte das Event manuell) Bit 7: < reserved > Bit 8: classic header (Ereignis wurde durch die TraceEvent-Funktion erzeugt (MOF-kodiert)) Bit 9: processor index (Prozessor-Index ist definiert)
USHORT	EventProperty	Informationen zum Dekodieren der Event-Daten 2: Datenstrukturbeschreibung aus registriertem XML-Manifest 1: Datenstrukturbeschreibung aus mitgelieferten XML-Manifest .....(am Anfang der Event-Daten) 0: Datenstrukturbeschreibung aus registrierter MOF-Klasse

Tabelle 3.9: Feldes des ETW-Ereignis-Headers (Teil 1) [Cha20c][Mic5c][Mic5d][Mic5e]

GUID	ProviderId	GUID der ETW-Provider-Klasse
USHORT	Id	ID des Eventtyps (Eindeutig innerhalb der ETW-Provider-Klasse)
UCHAR	Version	Version des Eventtyps (verschiedene Versionen können unterschiedliche Datenstrukturen besitzen)
ULONG	ProcessId	Prozess-ID des Prozesses, der das Ereignis erzeugt hat
ULONG	ThreadId	Thread-ID des Threads, der das Ereignis erzeugt hat
LARGE_INT	TimeStamp	Zeitpunkt, zu dem das Ereignis erzeugt wurde
ULONG	KernelTime	Kernel-Zeit-Zeitraum zwischen Thread-Start und Ereignisauslösung (nur wenn Bit 4 der Flags gesetzt ist)
ULONG	UserTime	User-Zeit-Zeitraum zwischen Thread-Start und Ereignisauslösung (nur wenn Bit 4 der Flags gesetzt ist)
ULONG64	ProcessorTime	Prozessor-Zeit-Zeitraum zwischen System-Start und Ereignisauslösung (nur wenn Bit 4 der Flags nicht gesetzt ist)
UCHAR	Channel	Windows-Event-Log-Channel, dem das Ereignis zugeordnet werden soll, falls es in den Windows-Event-Log geschrieben wird. .....0: Manifest-based Event (normaly used) ....11: TaceLogging-based Event (MOF) (normaly used) other: undocumented or provider class specific
UCHAR	Level	Log-Level (Verbose, Info, Warning, Error, Critical)
UCHAR	Opcode	Provider-Klassen-spezifischer Subtyp
USHORT	Task	Provider-Klassen-spezifische Zusatzinformation
ULONGLONG	Keyword	Bit-Maske spezifiziert, welchen Keywords das Ereignis zugeordnet wird. Die Keywords sind Provider-Klassen-spezifisch.

Tabelle 3.10: Feldes des ETW-Ereignis-Headers (Teil 2) [Cha20c][Mic5c][Mic5d][Mic5e]

Listing 3.2: ETW Event Header (Quelle: Microsoft API-Dokomentation [Mic5d])

```
typedef struct _EVENT_HEADER {
    USHORT           Size;
    USHORT           HeaderType;
    USHORT           Flags;
    USHORT           EventProperty;
    ULONG            ThreadId;
    ULONG            ProcessId;
    LARGE_INTEGER    TimeStamp;
    GUID             ProviderId;
    EVENT_DESCRIPTOR EventDescriptor;
    union {
        struct {
            ULONG KernelTime;
            ULONG UserTime;
        } DUMMYSTRUCTNAME;
        ULONG64 ProcessorTime;
    } DUMMYUNIONNAME;
```

```
18     GUID             ActivityId;
19 } EVENT_HEADER, *PEVENT_HEADER;
20
21 typedef struct _EVENT_DESCRIPTOR {
22     USHORT     Id;
23     UCHAR      Version;
24     UCHAR      Channel;
25     UCHAR      Level;
26     UCHAR      Opcode;
27     USHORT     Task;
28     ULONGLONG Keyword;
29 } EVENT_DESCRIPTOR, *PEVENT_DESCRIPTOR;
```

Der Datenbereich mit den Parametern ist in einem zum Ereignistyp passenden binären Format serialisiert abgelegt. Der Ereignistyp ist durch die Kombination aus ETW-Provider-Klassen-GUID, Event-ID und Event-Version eindeutig spezifiziert. Alle drei Attribute sind über den Header kodiert. Um die Daten eines Ereignisses zu interpretieren, ist eine beschreibende Struktur notwendig. Diese ist durch die Provider-Klasse gegeben.

3.6.8 ETL-Protokolldateien

Wie beschrieben, können ETW-Sessions so konfiguriert werden, dass ihre Event-Objekte automatisch vom ETW-System im Kernel in Log-Dateien geschrieben werden und somit auf einem Datenträger persistent gespeichert werden, um zu einem späteren Zeitpunkt von einer ETW-Consumer-Anwendung verarbeitet zu werden. Dazu sammelt das ETW-System eine größere Menge Event-Objekte in den Puffern der ETW-Session und schreibt diese dann gebündelt zu günstigen Zeitpunkten effizient auf den Datenträger, sodass der Einfluss auf den restlichen Betrieb des Systems minimiert wird. Zuständig dafür ist der ETW-Logging-Thread der ETW-Session [AIRS21, S. 511ff]. Event-Tracing-for-Windows (ETW) nutzt zur persistenten Protokollierung auf Datenträgern ein eigenes Dateiformat, welches die Eigenschaft aufweist, dass ETW-Event-Daten schnell geschrieben und gelesen werden können. Das von ETW benutzte Dateiformat trägt den Namen „Event Trace Log“ (ETL). Bei ETL handelt es sich um ein Binärformat, das nicht öffentlich dokumentiert ist [AIRS21, S. 512ff][PL20]. Zudem gibt es außerhalb der ETW-API keine offizielle Software-Bibliothek, um das ETL-Format zu interpretieren und somit ETL-Dateien zu lesen. Das ETW-System selbst enthält die Logik zum Schreiben der Event-Daten im ETL-Format im Kernel. Der spätere Zugriff durch eine Analyseanwendung auf diese speziellen Log-Dateien ist offiziell nur über entsprechende Funktionen der ETW-Consumer-API vorgesehen. Anwendungen müssen sich mit dieser API beim ETW-System im Kernel als ETW-Consumer anmelden und können dann über die API Event-Objekte aus der

bei der Anmeldung spezifizierten ETL-Datei auslesen [MicAa]. Alle ETW-Consumer-Anwendungen verarbeiten ETL-Datein über diesen Weg. Dies hat zur Folge, dass ETL-Dateien nur auf Systemen gelesen werden können, welche die ETW-Consumer-API bereitstellen. Diese ist Teil des Windows-Betriebssystems. Die Einschränkungen sind jedoch noch stärker, so können die Daten aus ETL-Dateien nur sicher auf dem Windows-System korrekt interpretiert werden, auf dem diese auch erstellt wurden, da für das Dekodieren der in den Dateien gespeicherten Event-Objekten die Strukturinformationen bezüglich den Formaten der Ereignistypen (Attribute-Felder, Datentypen usw.) benötigt werden [AIRS21, S. 513ff]. Diese stehen in Form der installierten Provider-Klassen nicht in jedem Fall auf einem anderen Windows-System zur Verfügung. Microsoft sieht ETL-Dateien auch nicht als portable Log-Dateien vor, sondern als eine Schnittstelle zwischen ETW-Session und ETW-Consumer, welche die besondere Eigenschaft aufweist, dass die Daten persistent auf einem Datenträger gespeichert werden können. Sollen die ETW-Event-Daten als portable Protokollierung weitergegeben und auf anderen Systemen analysiert werden, soll die Consumer-Anwedung die Event-Daten in ein eigenes Format wandeln und in diesem weitergeben. Microsoft empfiehlt die eigene Windows-Event-Log-Library (`WevtApi.dll`). Diese kann ETW-Events über `SecHost.dll` als ETW-Consumer aus einer ETL-Datei beziehen und mittels `TDH.dll` die serialisierten Event-Daten dekodieren. Die Windows-Event-Log-Library erzeugt die Events als XML-Datensatz (XPath 1.0) und kann diese sowohl dem Windows-Event-Log hinzufügen, als auch in komprimierten Windows-Event-Log-Dateien im EVTX-Format speichern. Letztere enthalten als portables Dateiformat alle Informationen, um auf anderen Systemen verarbeitet werden zu können [AIRS21, S. 513ff][MicAb].

In manchen Fällen ist das Lesen von ETL-Daten auch ohne eine ETW-Consumer-Anwendung auf der zu überwachenden Maschine erwünscht. Das Airbus-CERT hat sich dieser Problematik in einem Projekt gewidmet und eine Python-Software entwickelt, die es möglich machen soll, ETL-Dateien ohne die ETW-API einzulesen und die Event-Daten zu interpretieren. Die Software kann auch auf anderen Betriebssystemen wie z.B. Linux ausgeführt werden und weist keine Abhängigkeiten zum ETW-System auf. Dazu wurde das ETL-Dateiformat reverse-engineered. Um die Event-Objekte korrekt zu interpretieren, bringt die Python-Software eine große Menge von Dekodern für verschiedene unter Windows standardmäßig vorhandener Provider-Klassen mit. Somit kann nicht allgemein jeder ETL-Datensatz verarbeitet werden. Solange die protokollierten Event-Objekte jedoch auf den unterstützten Provider-Klassen basieren, sollte das Dekodieren funktionieren [PL20].

ETL-Dateien beginnen mit einem binären Header, der die ETW-Session, auf der die Protokollierung beruht, beschreibt. Darauf folgen die binär kodierten Event-Objekte nacheinander. Die Event-Objekte besitzen in etwa das gleiche serialisierte Binärformat, das sie auch aufweisen, wenn sie sich im Puffer der ETW-Session befinden (`EVENT_RECORD`).

Jedes Event-Objekt besitzt seinen eigenen Event-Header am Anfang des Daten-Blobs des Event-Objektes. Ein Teil dieses Headers ist für jedes ETW-Event gleich aufgebaut. Aus diesem ist unter anderem die GUID der Provider-Klasse zu entnehmen. Je nachdem, um welchen Klassentyp und welche Windows-Version es sich handelt, variiert der zweite Teil des Headers. Auch dieser ist jedoch noch direkt zu lesen, wenn die Datenstruktur bekannt ist [Cha20c]. Nach dem Header folgen die Nutzdaten des Event-Objektes. Diese enthalten die Werte aller Event-Attribute. Ihre Struktur ist zum einen abhängig vom Ereignistyp, zum anderen sind die Nutzdaten je nach Klassentyp unterschiedlich kodiert. Die Nutzdaten eines Events, dass auf einer Manifest-basierten Event-Provider-Klasse basiert, sind z.B. anders kodiert als die Nutzdaten eines Events einer MOF-Provider-Klasse. Der Ereignistyp eines Event-Objektes ist über das Header-Feld `ID` spezifiziert. Die Strukturdefinition der Nutzdaten ist unter der Event-ID in der Provider-Klasse spezifiziert, auf der das Event-Objekt basiert [PL20][AIRS21, S. 513ff].

Für eine Buffered-ETW-Session kann die ETW-Controller-Anwendung den Speicherort und Bezeichner der ETL-Datei spezifizieren, in welche die Event-Daten geschrieben werden sollen. Dieser kann sich in einem frei wählbaren Verzeichnis auf jedem logischen Laufwerk befinden. Standardmäßig empfiehlt Microsoft das Verzeichnis `C:\PerfLogs\`. Für das Protokollieren kann aus verschiedenen Schreib-Modi gewählt werden. So kann z.B. eine maximale Dateigröße festgelegt werden, ab welcher die Datei wie ein Ring-Puffer wieder überschrieben wird (Circular-Mode). Das ETW-System konfiguriert den Security-Deskriptor der ETL-Datei und dessen Access-Control-List (ACL) so, dass diese dem Security-Deskriptor des ETW-Session-Objektes entspricht. Auf eine ETL-Datei haben somit dieselben Benutzerkonten die gleichen Zugriffsberechtigungen, wie auf die ETW-Session, auf der die ETL-Protokolldatei basiert [Mic9a].

Neben der durch den Controller für eine Buffered-ETW-Session spezifizierten ETL-Datei, legt das ETW-System intern auch für jede aktive Real-Time-ETW-Session eine temporäre ETL-Datei an. Dies geschieht auch für Real-Time-Sessions, die ausdrücklich so konfiguriert sind, dass die Event-Objekte nur direkt an eine Consumer-App weitergeleitet werden und nicht in eine Datei geschrieben werden sollen. Die temporären ETL-Dateien befinden sich unter `%SystemRoot%\System32\LogFiles\WMI\RtBackup`. Der Dateiname dieser Dateien wird automatisch generiert und setzt sich aus dem Präfix „`EtwRT`“ und dem Bezeichner der ETW-Session zusammen. Das ETW-System speichert in bestimmten Fällen in diesen Dateien temporär Event-Objekte, bevor sie an eine Consumer-Anwendung geliefert werden. So wird es möglich, eine Consumer-Anwendung erst kurze Zeit nach dem Start der ETW-Session als Real-Time-Consumer für diese Session zu registrieren und trotzdem noch alle Event-Objekte zu beziehen. Ist ein Puffer der Session voll, muss dessen Inhalt spätestens an den Consumer weitergereicht werden, damit dieser nicht verloren geht. Ist noch kein Consumer-Prozess registriert oder nicht bereit, werden die Event-Objekte in die temporäre Datei geschrieben. Dieser Me-

chanismus garantiert nicht, dass Event-Objekte nicht verloren gehen können, wenn die Consumer-Anwendung zu lange nicht reagiert oder die eingehende Event-Menge zu groß ist. Unter anderem kann dieser Mechanismus benutzt werden, wenn eine Auto-Logger-Session zur Protokollierung des Bootvorgangs als Real-Time-Session spezifiziert wird. Die Consumer-App kann in diesem Fall erst nach der abgeschlossenen System-Initialisierung starten und trotzdem unter Umständen noch alle während des Bootvorgangs gesammelten Event-Objekte direkt aus der Session (Real-Time-Betrieb) erhalten [AIRS21, S. 511ff].

3.6.9 Ereigniserzeugung (Provider)

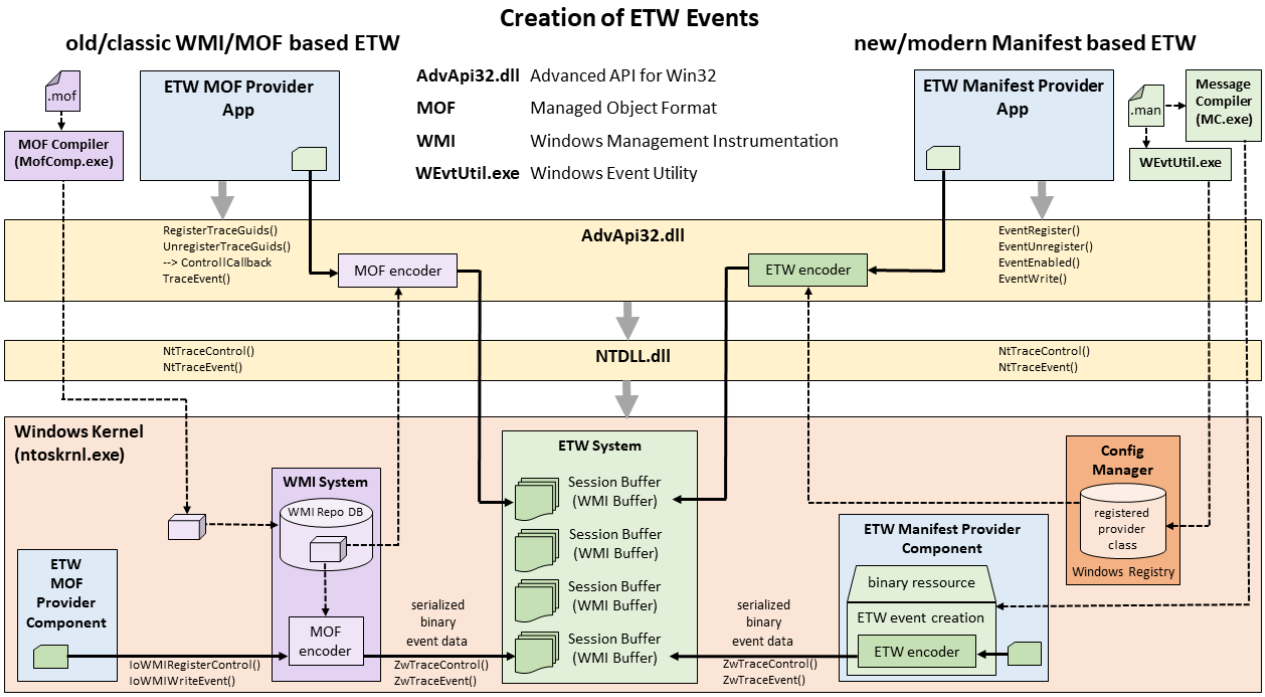

Abbildung 3.2: Ereigniserzeugung

Eine ETW-Provider-Komponente muss sich beim ETW-System anmelden, um Ereignisse bereitstellen zu können. Dieser Vorgang wird als Provider-Registrierung bezeichnet. Da dieselbe Provider-Komponente (Code-Komponente) wie beschrieben in mehreren Prozessen ausgeführt werden kann, kann ein ETW-Provider mehrfach parallel am ETW-System angemeldet bzw. registriert sein. Bei der Anmeldung muss die ETW-Provider-Klasse festgelegt werden, für die Ereignisse erzeugt werden sollen. Die ETW-Provider-Klasse muss auf dem System installiert sein und wird über ihre GUID adressiert. Sie spezifiziert die ETW-Ereignistypen, die der Provider erstellen kann. Das ETW-System verwaltet intern für jede ETW-Provider-Klasse, für die mindestens eine Provider-Registrierung stattgefunden hat, eine Liste mit Prozessen, die sich als Provider-Prozesse für diese Klasse

angemeldet haben. Nachdem die Anmeldung beim ETW-System durchgeführt wurde, kann die ETW-Provider-Komponente Ereignisse erstellen und diese an das ETW-System übermitteln [Mic8a][AIRS21, S. 509ff].

Es existieren vier verschiedene Arten von ETW-Provider-Klassen. Diese werden bezeichnet als „Manifest-Provider-Klassen“, „Trace-Provider-Klassen“, „MOF-Provider-Klassen“ und „WPP-Provider-Klassen“. Die Unterschiede sind im Abschnitt „ETW-Provider-Klassen“ detailliert erklärt. Je nachdem auf welchem Klassentyp der ETW-Provider basiert, unterscheidet sich die zu verwendende API für die Registrierung und Event-Erzeugung. Im Folgenden ist der Vorgang für Manifest-basierte Provider und für MOF-Provider beschrieben. Zudem variiert die API zwischen User-Mode-Providern und Kernel-Mode-Providern.

3.6.9.1 Event-Erzeugung: User-Mode-Manifest-Provider

Ein User-Mode-Manifest-Provider kann laut der Microsoft-Dokumentation in zwei unterschiedlichen Varianten realisiert werden. Zum einen kann die Windows-API zur Erstellung und Serialisierung des Event-Objektes benutzt werden. Alternativ kann speziell für die Erzeugung und Serialisierung der Event-Objekte auch ein spezieller ETW-Compiler Code erzeugen, der statisch als binäre Ressource in das Executable gelinkt werden kann. Der hinzugebundene Code exportiert Funktionen, die alternativ zur Windows-API benutzt werden können, um Event-Objekte zu erzeugen [Mic8b].

Als erstes wird der Erzeugungsvorgang über die Windows-API beschrieben. Einem Manifest-basierten Provider stellt die Windows-API im User-Mode für die An- und Abmeldung die Funktionen `EventRegister` und `EventUnregister` bereit. Über diese kann der Prozess sich als ETW-Provider für eine bestimmte installierte Provider-Klasse anmelden, deren GUID als Argument der `EventRegister`-Funktion übergeben werden muss. Beide Funktionen befinden sich in der `AdvAPI32.dll` (Advanced Win32-API). Ihre Deklarationen können über die Header-Datei `evntprov.h` eingebunden werden. Optional, kann bei der Registrierung eine Callback-Funktion spezifiziert werden, die von der Windows-API aufgerufen wird, wenn die Provider-Klasse aktiviert oder deaktiviert wird. Dies ermöglicht dem Provider, Vorgänge durchzuführen, die bei einer Aktivierung der Provider-Klasse notwendig sein können, wie z.B. Initialisierungen oder das Erstellen von DC- oder Rundown-Event-Objekten (werden im Abschnitt „DC- und Rundown-Ereignisse“ näher erläutert). Eine Provider-Klasse wird dann aktiviert, wenn sie von mindestens einer ETW-Session abonniert ist. Die API-Funktionen `EventRegister` und `EventUnregister` verwenden die Funktion `NtTraceControl` der `NTDLL.dll` mit verschiedenen Op-Codes, um die Registrierungsvorgänge dem ETW-System im Kernel mitzuteilen. `NtTraceControl`

führt den Systemcall durch, um mit dem ETW-System im Kernel zu kommunizieren. Bevor ein ETW-Event erzeugt wird, soll über die Funktion `EventProviderEnabled` und/oder `EventEnabled` geprüft werden, ob die Provider-Klasse aktiviert ist und der zu erzeugende Eventtyp protokolliert werden soll (Filterung). Die Windows-API hält den Zustand vor, sodass für diese Abfragen keine Systemcalls notwendig sind. Die ETW-User-Mode-Bibliothek wird vom ETW-System im Kernel benachrichtigt, wenn der Aktivierungszustand für die jeweilige Provider-Klasse, unter der die Provider-Komponente angemeldet ist, sich ändert. Auch diese Vorgänge werden zwischen dem ETW-System im Kernel und der ETW-API der `AdvAPI32.dll` über die Funktion `NtTraceControl` der `NTDLL.dll` abgewickelt. Für die eigentliche Erzeugung, Serialisierung, Kodierung und Übermittlung eines Event-Objektes an das ETW-System stellt die Windows-API die vier Funktionen `EventWrite`, `EventWriteEx`, `EventWriteString` und `EventWriteTransfer` zur Verfügung. Je nach Art der Eventdaten ist eine andere der vier Funktionen zu verwenden. Auch diese Funktionen befinden sich in der `AdvAPI32.dll`-Bibliothek und können über die Header-Datei `evntprov.h` eingebunden werden. Für normale Ereignistypen, deren Strukturen (Aufbau, Attribute) in der Provider-Klasse spezifiziert sind, werden die Funktionen `EventWrite` und `EventWriteEx` verwendet. Die Event-Attribute für das zu erzeugende Event-Objekt werden den Funktionen in einer speziellen Weise als Parameter übergeben. Allgemeine Daten des Event-Headers müssen in eine anzulegende Struktur vom Typ `EVENT_DESCRIPTOR` eingetragen werden. Dies sind die Event-ID, die Event-Version, der Event-Log-Kanal, der Log-Level, der Op-Code und eine Bitmaske, welche spezifiziert, welchen Filter-Keywords das Event angehört. Diese beschriebenen Daten-Felder sind für alle ETW-Events Ereignistyp-übergreifend vorhanden und befinden sich im allgemeinen Header jedes ETW-Event-Objektes. Den Funktionen wird ein Adresszeiger auf die erstellte `EVENT_DESCRIPTOR`-Struktur übergeben. Die speziellen Attribute, die Ereignistyp-spezifisch sind, werden den Funktionen `EventWrite` und `EventWriteEx` durch einen Zeiger auf ein Array mit dem Element-Typ `EVENT_DATA_DESCRIPTOR` übergeben. Das Array ist quasi eine Liste von Event-Data-Deskriptoren. Jedes Data-Deskriptor-Element enthält einen Adresszeiger auf die Daten eines Attributes des zu erzeugenden Event-Objektes. Die Position des Daten-Deskriptors eines Attributfeldes in der Deskriptor-Liste muss mit der Position der Spezifikation des Attributfeldes im Instrumentierungs-Manifest der Provider-Klasse übereinstimmen. Der Windows-API-Code in der `AdvAPI32.dll` serialisiert und kodiert die Event-Daten zu einem Event-Objekt, dass dann an das ETW-System im Kernel weitergereicht wird. Die für die Serialisierung und Kodierung notwendige Spezifikation des Ereignistyps der Provider-Klasse lädt die `AdvAPI32.dll` aus der Windows-Registry, in der die Provider-Klasse durch ihre Installation registriert wurde. Für die Übertragung des Event-Objektes zum ETW-System in den Kernel hinein wird die Funktion `NtTraceEvent` der `NTDLL.dll` benutzt. Diese gibt das serialisierte und kodierte Event-Objekt über einen Systemcall an das ETW-System weiter, welches das Objekt in den Puffern verschiedener ETW-Sessions speichert, die das entsprechende Ereignis

abonniert haben. Das bei Manifest-basiertem ETW verwendete Serialisierungsformat ist nicht dokumentiert, ebenso ist die Kodierungs-Logik innerhalb der `AdvAPI32.dll` als Closed-Source-Software nicht öffentlich einsehbar [Mic8b].

Wie erwähnt müssen nicht die Funktionen der Windows-API benutzt werden. Stattdessen kann mit der Windows-Anwendung „Message Compiler“ (`MC.exe`) Maschinencode für die Erzeugung der Event-Objekte generiert werden. Der Compiler bekommt als Argument den Pfad zum Instrumentierungs-Manifest (.man-Datei) der zu verwendenden Provider-Klasse. Mit den darin enthaltenen Informationen bezüglich der Struktur der Event-Objekte für die verschiedenen Ereignistypen erzeugt der Compiler den Code für die Serialisierung und Kodierung der Event-Objekte. Zusätzlich zur binären Ressource (.bin) mit dem Code, der statisch in das Executable zu linken ist, werden auch Header-Dateien mit den exportierten Funktionsdeklarationen und weiteren Macros erzeugt, um den generierten Code im eigenen Software-Entwicklungs-Projekt zu verwenden [Mic2e][Mic8d]. Das durch den Message-Compiler (`MC.exe`) erzeugte Modul bietet in etwa die gleiche Funktionalität wie die ETW-Provider-Funktionen der Windows-API. Eine Funktion `McGenEventRegister` meldet den Prozess als Provider beim ETW-System an. Die Provider-Klasse muss in diesem Fall nicht spezifiziert werden, da der Code von `McGenEventRegister` bereits exklusiv für eine spezielle Provider-Klasse erzeugt wurde. Ebenso wie die Windows-API bietet `McGenEventRegister` die Funktionalität, eine Callback-Funktion zu registrieren, um auf Aktivierungen und Deaktivierung der Provider-Klasse zu reagieren. Ebenso kann und sollte vor der Erzeugung eines ETW-Event-Objektes geprüft werden, ob die Provider-Klasse und der Entsprechende Ereignistyp aktiviert sind bzw. von mindestens einer Session abonniert sind. Der vom Message-Compiler (`MC.exe`) generierte Code ruft intern, um die Anmeldung umzusetzen, ebenso wie die Windows-API die Funktion `NtTraceControl` der `NTDLL.dll` auf. Diese führt den Systemcall zur Kommunikation mit dem ETW-System im Kernel durch. Die Erzeugung des Event-Objektes geschieht mit der Funktion `McGenEventWrite`, die das Gegenstück zu der Funktion `EventWrite` der Windows-API darstellt. `McGenEventWrite` implementiert die Serialisierung und Kodierung der übergebenen Event-Daten zu einem Event-Objekt. Die benötigten Spezifikationen der Ereignistypen der Provider-Klasse müssen in diesem Fall jedoch nicht aus der Registry gelesen werden, da die Serialisierungs- und Kodierungs-Logik speziell für die Ereignistypen der Provider-Klasse durch den Message-Compiler (`MC.exe`) erzeugt wurden. Auch die `McGenEventWrite`-Funktion verwendet intern die Funktion `NtEventWrite` der `NTDLL.dll`, welche den Systemcall durchführt, um die serialisierten Daten des Event-Objektes dem ETW-System im Kernel zu übergeben [Mic8d].

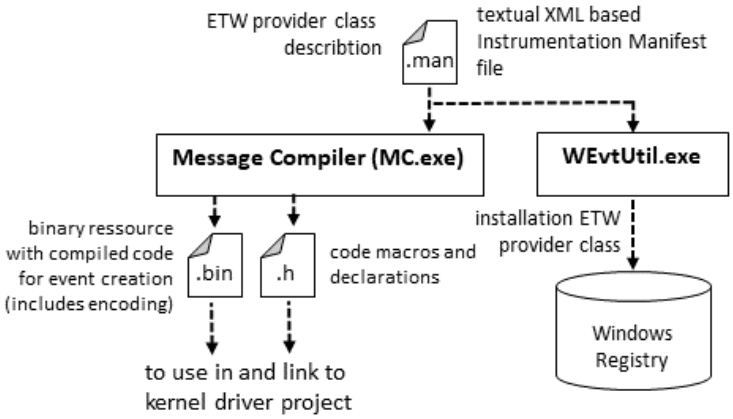

Abbildung 3.3: Erzeugung von Provider-Code mittels des Message-Compilers

3.6.9.2 Event-Erzeugung: User-Mode-MOF-Provider

Für einen MOF-basierten Provider gestaltet sich der Vorgang sehr ähnlich, jedoch sind die zu verwendenden API-Funktionen andere. Zudem besteht nur ein Verfahren über die Windows-API. Eine Möglichkeit generierten Code eines Message-Compilers (`MC.exe`) zu nutzen, besteht für MOF-Provider nicht. Zur An- und Abmeldung eines MOF-Providers stellt die Windows-API die Funktionen `RegisterTraceGuidsA/W` und `UnregisterTraceGuids` bereit. Alle Funktionen befinden sich ebenfalls in der `AdvAPI32.dll`. Jedoch sind diese statt in `evntprov.h` in der Header-Datei `evntrace.h` deklariert. Beim Anmelden soll eine Callback-Funktion registriert werden, die aufgerufen wird, wenn sich der Aktivierungszustand der Provider-Klasse ändert. Anders als beim zuvor beschriebenen Vorgehen für Manifest-basierte Provider, muss der Anwendungscode des MOF-Providers den Aktivierungszustand der Provider-Klasse, auf der er basiert, selbst mitverfolgen. Änderungen bezüglich des Aktivierungszustandes zu erkennen, ist für den Anwendungscode nur über die Signalisierung durch die Callback-Routine möglich. Die An- und Abmelde-Funktionen sowie die Funktionen des Signalisierungs-Mechanismus der ETW-API in der `AdvAPI32.dll` verwenden alle für die Kommunikation mit dem ETW-System im Kernel die Funktion `NtTraceControl` der `NTDLL.dll`. Für das Übermitteln eines MOF-basierten Event-Objektes stehen in der `AdvAPI32.dll` über den Header `evntrace.h` die Funktionen `TraceEvent` und `TraceEventInstance` zur Verfügung. Je nach Art des zu übertragenen Events soll eine der beiden Funktionen benutzt werden. Der `TraceEvent`-Funktion wird als Parameter ein Adresszeiger vom Typ `VOID`-Pointer auf eine selbst zu definierende Datenstruktur eines selbst zu definierenden Struktur-Typs (z.B. `MY_EVENT`) übergeben. Die Struktur muss jedoch nach speziellen Regeln aufgebaut sein. Das erste Feld der Struktur muss den Typ `EVENT_TRACE_HEADER` aufweisen und soll die allgemeinen Header-Informationen des zu erzeugenden Event-Objektes enthalten. Darauf folgt als zweites Feld ein Array vom Element-Typ `MOF_FIELD`. Die Länge des Arrays ist die Anzahl an Attributen, die der entsprechenden Ereignistyp aufweist. Jedes Ele-

ment dieses Arrays spezifiziert einen Attribut-Wert des zu erstellenden Event-Objektes. Die `MOF_FIELD`-Struktur enthält zwei Felder. Das erste ist ein Adresszeiger auf die Daten des Attribut-Wertes. Das zweite Feld gibt die Länge des Attribut-Wertes in Bytes an. Der Code der Funktionen `TraceEvent` und `TraceEventInstance` aus der `AdvAPI32.dll` serialisiert und kodiert die übergebenen Event-Daten zu einem Event-Objekt. Dabei wird ein MOF-Encoder eingesetzt, der auf Basis von MOF-Klassen MOF-Objekte erzeugt. Das Verfahren sowie das Serialisierungsformat sind nicht öffentlich dokumentiert. Die MOF-Klassen, welche die Struktur der Ereignistypen der Provider-Klasse beschreiben, sind in der Repo-Datenbank des WMI-Systems durch ihre Installation registriert. Die `AdvAPI32.dll`-Bibliothek lädt diese Informationen aus der WMI-Repo-DB, um das binäre Event-Objekt zu erzeugen. Danach wird dieses an das ETW-System im Kernel übergeben. Für die Übertragung des Event-Objektes zum ETW-System in den Kernel hinein verwendet die ETW-API in der `AdvAPI32.dll` die Funktion `NtTraceEvent` der `NTDLL.dll`, die den Systemcall durchführt, um die serialisierten Daten zu übertragen [Mic8c].

3.6.9.3 Event-Erzeugung: Kernel-Mode-Manifest-Provider

Einer Kernel-Komponente, die als Manifest-Provider ETW-Ereignisse erzeugen möchte, stehen ebenfalls Methoden bereit, die Microsoft in der Dokumentation für Treiber-Entwickler beschreibt. Diese Vorgehensweisen sind den beschriebenen Verfahren für den User-Mode recht ähnlich. Anders als im User-Mode steht im Kernel-Mode keine ETW-API zur Verfügung, welche die Serialisierung und Kodierung der ETW-Event-Daten umsetzt, sodass im Kernel nur der Weg über den speziell erzeugten Provider-Code des Message-Compilers (`MC.exe`) gewählt werden kann. Der Treiber-Entwickler muss dabei mithilfe des von ihm erstellten Instrumentierungs-Manifestes (.man-Datei), das die Provider-Klasse beschreibt, durch den Message-Compiler (`MC.exe`) Maschinencode erzeugen lassen, der statisch in das Executable des Kernel-Moduls (SYS-Datei) gelinkt wird. Dieser generierte Code stellt Methoden für die Registrierung und Ereigniserzeugung bereit. Mittels `McGenEventRegister` kann sich der Provider für seine Provider-Klasse anmelden. Es existieren ähnliche Mechanismen bezüglich der Erkennung des Aktivierungszustandes der Provider-Klasse wie im User-Mode. Das eigentliche Erzeugen, Serialisieren, Kodieren und Übermitteln des Event-Objektes an das ETW-System geschieht über die Funktion `McGenEventWrite` in gleicher Weise, wie es für den User-Mode beschrieben wurde [Mic8d]. Die Funktionen des statisch gelinkten Codes des Message-Compilers verwenden die Kernel-internen Funktionen `ZwTraceEvent` und `ZWTraceControl` (in `ntoskrnl.exe`), um mit dem ETW-System zu kommunizieren. Diese Funktionen entsprechen den Funktionen `NtTraceControl` und `NtTraceEvent`, die ETW-Provider aus dem User-Mode verwenden [TKG21a, S. 22].

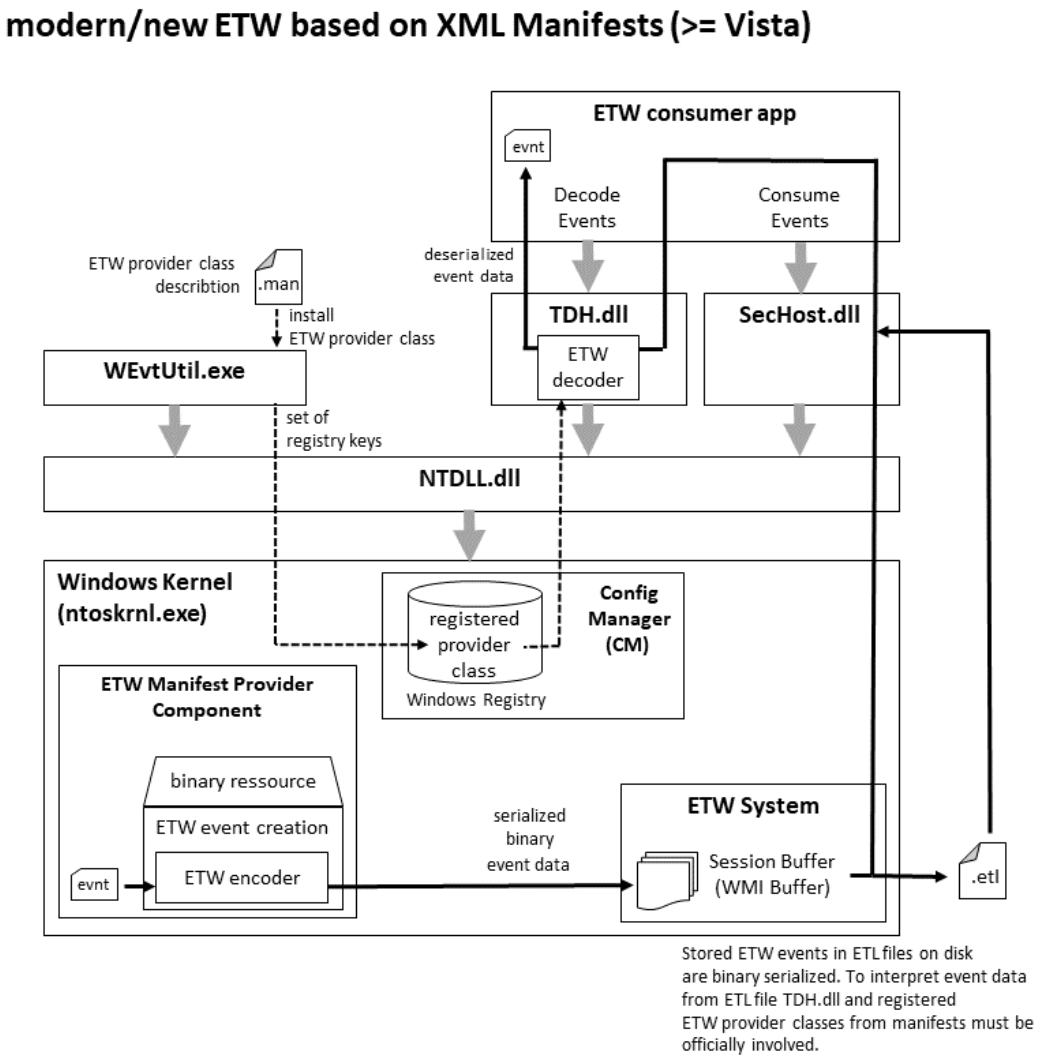

Abbildung 3.4: Erzeugung von Events durch Manifest-basierte Provider

3.6.9.4 Event-Erzeugung: Kernel-Mode-MOF-Provider

Für MOF-basierte Kernel-Provider stellt das Windows-Driver-SDK (WDM-SDK: Windows Driver Modell SDK) eine Reihe von Funktionen für die Erzeugung von ETW-Events auf Basis von MOF-Provider-Klassen bereit, die allesamt in der Header-Datei `wdm.h` deklariert sind und direkt vom Windows-Kernel selbst (`ntoskrnl.exe`) exportiert werden. Die Funktion `IoWMIRegistrationControl` bietet verschiedene Funktionalitäten, die über einen Actions-Op-Code, der als Parameter übergeben wird, spezifiziert werden können. Unter anderem kann über diese Funktion die Kernel-Komponente sich als MOF-basierter ETW-Provider beim ETW-System anmelden und abmelden. Die Funktion übermittelt dem WMI-System eine Anfrage für eine ETW-MOF-Provider-Anmeldung. Als Antwort enthält der Treiber ein IRP-Packet über die IRP-Treiber-Schnittstelle, welche die Datenstruktur `WMIREGGUID` enthält. In dieser ist die GUID der MOF-Provider-Klasse

zu spezifizieren, für die sich der Provider beim ETW-System anmeldet. Zudem kann über eine Bitmaske (`WMIREG_FLAG_TRACE_CONTROL_GUID`) innerhalb der Struktur eingestellt werden, welche Ereignistypen der Provider-Klasse diese Provider-Komponente erzeugen kann. Um ein ETW-Event zu erzeugen, zu serialisieren, zu kodieren und das binäre Event-Objekt an das ETW-System zu übertragen, bietet der Kernel die Funktion `IoWMIWriteEvent`. Der Funktion werden die Event-Daten über einen Adresszeiger auf eine `WNODE_EVENT_ITEM`-Struktur, die die Event-Daten beschreibt, übergeben. Der hinter der Funktion `IOWMIWriteEvent` befindliche Kernel-Code des WMI-Systems sorgt für die Serialisierung und Kodierung der Event-Daten zu einem binären und gepackten Event-Objekt. Das binäre MOF-Serialisierungsformat ist nicht öffentlich dokumentiert. Die für die Serialisierung und Kodierung notwendige Strukturbeschreibung der Ereignistypen wird den im WMI-System registrierten MOF-Klassen der Provider-Klasse entnommen. Das WMI-System gibt das Event-Objekt an das ETW-System weiter, welches das Objekt in die Puffer entsprechenden ETW-Session einfügt [Mic8e].

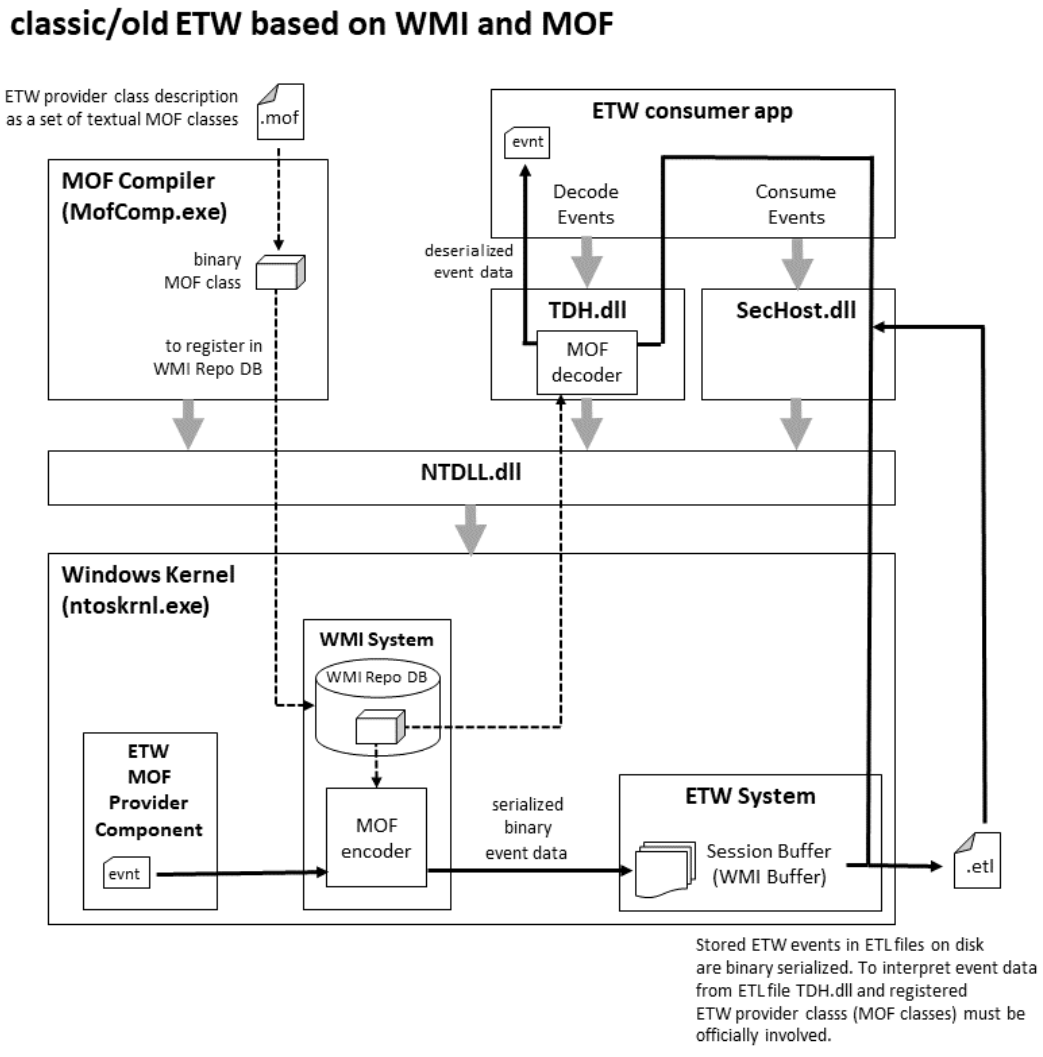

Abbildung 3.5: Erzeugung von Events durch MOF-basierte Provider

Wie das WMI-System und das ETW-System Kernel-intern kommunizieren ist nicht öffentlich dokumentiert. Eventuell werden für die Kommunikation bezüglich der Registrierung von MOF-Providern und der Übertragung der serialisierten Event-Objekte auch die Funktionen `ZwTraceControl` und `ZwTraceEvent` benutzt. Dies ist aber nur eine Vermutung und bedarf weiterer Untersuchungen.

3.6.9.5 Event-Erzeugung: Windows-interne Kernel-Provider

Die beschriebenen Vorgänge im Kernel entsprechen den Anleitungen der Windows-Dokumentation für Treiber-Entwickler. Es ist zu vermuten, dass Microsoft für die eigenen Provider an den Basisfunktionen des Windows-Kernels dieselbe Methodik und dieselben exportierten Funktionen nutzt. Es besteht aber auch die Möglichkeit, dass die internen Kernel-Provider andere nicht dokumentierte Funktionen zur Provider-Registrierung, Event-Erzeugung, -Serialisierung, -Kodierung und -Übertragung nutzen.

3.6.10 Sessionverwaltung (Controller)

User-Mode-Anwendungen stellt die Windows-API Funktionen zur Verwaltung von ETW-Sessions zur Verfügung. Über diese können neue ETW-Sessions angelegt und konfiguriert werden. Zudem kann der Controller über die ETW-Controller-API die Ablaufverfolgung mit einer ETW-Session starten und stoppen. Ein wichtiger Bestandteil der Konfiguration einer ETW-Session stellt die Spezifikation der Ereignistypen dar, die in der ETW-Session gesammelt werden sollen. Dafür werden die Provider-Klassen gewählt und für diese Filter-Konfigurationen spezifiziert. Diese Konfiguration bezüglich der zu erhaltenen Ereignistypen kann während die Session gestartet und aktiv ist, geändert werden. Hinter diesem Abonnieren und Abbestellen von Provider-Klassen und Ereignistypen steht der Vorgang der Provider-Aktivierung und -Deaktivierung. Dieser wird vom ETW-System im Kernel als Resultat über die Nutzung einer Provider-Klasse über alle Sessions hinweg durchgeführt. Dabei werden alle angemeldeten Provider-Prozesse einer Provider-Klasse benachrichtigt, dass sie aktiviert oder deaktiviert sind und ab diesem Zeitpunkt Event-Objekte liefern sollen oder nicht. Zudem wird ihnen die Filter-Bitmaske mitgeteilt, aus der sie entnehmen können, welche Gruppen von Ereignistypen sie erzeugen sollen und welche nicht.

Vor Windows Vista war die Erstellung neuer ETW-Sessions jedem Benutzerkonto gestattet. Seit Windows Vista muss ein ETW-Controller-Prozess im Kontext eines Benutzerkontos ausgeführt werden, dass der Gruppe „Performance Log Users“ angehört. Alternativ dürfen Benutzer, die über erweiterte Administratorberechtigungen (elevated administrative privileges) verfügen, ETW-Sessions erstellen und verwalten. Intern erlaubt

der Windows-Kernel Dienstprozessen, die im Kontext der Benutzerkonten `LocalSystem`, `LocalService` und `NetworkService` ausgeführt werden, ebenfalls die Erstellung und Konfiguration von ETW-Sessions. Unabhängig davon kann ein Controller für eine neu erstellte Session Zugriffsberechtigungen für das Session-Objekt konfigurieren. Über die Access-Control-List (ACL) des Security-Deskriptors kann der Controller einstellen, welche Benutzerkonten und Gruppen die ETW-Session als Controller konfigurieren und steuern dürfen und welche als Consumer aus der ETW-Session Event-Objekte erhalten dürfen. Somit besitzt ein Controller-Prozess, für den die zuerst genannten Bedingungen gelten (z.B. Mitglied der Gruppe „Performance Log Users“) nicht per se Zugriff auf alle ETW-Sessions des System, sondern nur zu denen, zu denen der Zugriff über die ACL gewährt wird. Dies sind standardmäßig die ETW-Sessions, die er selbst angelegt hat [AIRS21, S. 522ff].

Wie erwähnt existieren zwei Typen von ETW-Sessions. Normale ETW-Sessions und spezielle ETW-System-Sessions. Im Folgenden wird das Vorgehen zum Anlegen, Starten und Konfigurieren beider Typen von ETW-Sessions erklärt.

3.6.10.1 Anlegen und steuern einer normalen ETW-Session

Um eine neue ETW-Session anzulegen, ist die Daten-Struktur `EVENT_TRACE_PROPERTIES` durch die Controller-Anwendung anzulegen und deren Felder mit Werten zu füllen. Die Felder der Struktur stellen die allgemeinen Konfigurationsparameter der ETW-Session dar wie beispielsweise Name, Typ, Puffergröße, Dateiname der Logdatei. Über die Funktion `StartTraceA/W` wird die ETW-Session angelegt und gestartet. Der Funktion wird ein Adresszeiger auf die spezifizierte Konfigurations-Struktur übergeben. Die Funktion liefert der Anwendung einen Handle für den Zugriff auf das neu angelegt ETW-Session-Objekt zurück. Sowohl die Struktur `EVENT_TRACE_PROPERTIES` wie auch die Funktion `StartTraceA/W` sind in der Header-Datei `evntrace.h` deklariert. Die Funktion wird ab Windows 8.1 von der Security-Host-Bibliothek (`SecHost.dll`) exportiert. Zuvor wurde die Funktion durch `AdvAPI32.dll` bereitgestellt. Neben dem Erstellen und Starten neuer ETW-Sessions können über die Funktion `StartTraceA/W` auch vorhandene ETW-Sessions gestartet werden, die aktuell nicht aktiv sind. Über ein Feld in der Konfigurationsstruktur `EVENT_TRACE_PROPERTIES` wird der Name der ETW-Session spezifiziert. Bei der Erstellung einer neuen Session ist es wichtig, dass der verwendete Name noch nicht von einer vorhandenen ETW-Session verwendet wird. Ansonsten wird die vorhandene ETW-Session gestartet und ein Handle zu dieser geliefert. Durch diese Möglichkeit können durch spezifizieren des Namens einer vorhandenen ETW-Session diese fremde Session geöffnet und gestartet werden, wenn der Controller-Prozess über die notwendigen Berechtigungen verfügt und die Session nicht bereits aktiv ist [Mic9a].

Nachdem die ETW-Session angelegt und gestartet ist, können Provider-Klassen als Ereignisquellen abonniert werden. Dazu bietet die ETW-Controller-API die drei Funktionen `EnableTrace`, `EnableTraceEx` und `EnableTraceEx2` an. Alle drei Funktionen werden sowohl für das Abonnieren, das Abbestellen und das ändern der Filter-Einstellungen verwendet. Die drei Funktionen sind in der Header-Datei `evntrace.h` deklariert. Bereitgestellt werden die ersten beiden durch `AdvAPI32.dll`. Letztere Funktion (`EnableTraceEx2`) wird von der Bibliothek `SecHost.dll` exportiert. Sie besteht erst seit Windows 8.1. `EnableTrace` wird für das Abonnieren der älteren MOF-Provider-Klassen benutzt. Dahingegen werden die neueren Manifest-basierten Provider-Klassen mittels `EnableTraceEx` und `EnableTraceEx2` abonniert. Die Dokumentation empfiehlt ab Windows 8.1 `EnableTraceEx2` zu verwenden, welche ein Update der Funktion `EnableTraceEx` darstellt, das mehr Optionen bietet, jedoch im Prinzip die gleiche Grund-Funktionalität bereitstellt. Auch in neueren Windows-Versionen steht aber `EnableTraceEx` aus Gründen der Abwärtskompatibilität weiterhin zur Verfügung. Die Funktion `EnableTrace` benötigt zum Abonnieren der MOF-Provider-Klasse fünf Argumente. Den Session-Handle der Session, für die die Provider-Klasse abonniert werden soll, die GUID der Provider-Klasse, die abonniert oder abbestellt werden soll, einen boolschen Parameter, der angibt, ob die Provider-Klasse abonniert oder abbestellt werden soll, die Filter-Bitmaske bezüglich Provider-Klassen-spezifischer Keywords und der Level-Wert (`Critical`, `Error`, `Warning`, `Info`, `Verbose`), der angibt, ab welchem Log-Level Ereignisse in der Session gesammelt werden sollen. Soll über die `EnableTrace`-Funktion die Filter-Konfiguration einer abonnierten Provider-Klasse aktualisiert werden, ist für der boolschen Parameter so zu wählen, dass das Abonnieren signalisiert wird. Die `EnableTrace`-Funktion sollte nur mit GUIDs von Provider-Klassen benutzt werden, die auf MOF basieren. Manifest-basierte Provider-Klassen können aber auch über diese Funktion abonniert werden, jedoch sind die Einstellungsmöglichkeiten dadurch limitiert. Das Aufruf-Interface der Funktionen `EnableTraceEx` und `EnableTraceEx2` sieht sehr ähnlich aus, wie das der `EnableTrace`-Funktion, mit dem Unterschied, dass bei diesen Funktionen nur die GUIDs von Manifest-basierten Provider-Klassen benutzt werden können. Weitere Unterschiede sind, dass der Control-Code-Parameter neben dem Abonnieren, Aktualisieren und Abbestellen zusätzlich noch das Kommando `CAPTURE_STATE` vorsieht, dass dazu führt, dass Status-Informationen bezüglich der Provider-Klasse als ETW-Event geliefert werden. Ansonsten existieren für die Konfiguration der Keyword-Filter die zwei Parameter `MatchAnyKeywords` und `MatchAllKeywords`. Die Bitmaske `MatchAnyKeywords` spezifiziert die Provider-Klassen-spezifischen Schlüsselwörter, bei denen das Event-Objekt einem der spezifizierten Schlüsselwörter zugeordnet sein muss, um in der Session protokolliert zu werden. Dahingegen muss das Event-Objekt allen Schlüsselwörtern zugeordnet sein, die über die Bitmaske `MatchAllKeywords` spezifiziert wurden, um in der Session protokolliert zu werden. Ein weiterer Parameter mit der Bezeichnung `Timeout` steuert einen Blockier-Mechanismus in den Funktionen

`EnableTraceEx` und `EnableTraceEx2`. Ist der Wert dieses Parameters Null, findet die Aktivierung möglicher Provider der Provider-Klasse asynchron statt, sodass der aufrufende Thread des Controller Prozesses weiter ausgeführt wird. Jeder andere Wert spezifiziert eine Wartezeit (Timeout), den die Funktionen maximal wartet, bis die Provider aktiviert sind. In diesem Fall wird der aufrufende Thread blockiert, bis alle registrierten Provider der Provider-Klasse die Aktivierung bestätigt haben oder bis die angegebene maximale Verzögerung erreicht ist. Die Wartezeit kann auch über die Konstante `INFINITE` so eingestellt werden, dass in jedem Fall auf die Bestätigung aller registrierten Provider gewartet wird. Beide Funktionen bieten zudem über den Parameter `EnableParameters` optional weitere Konfigurationsmöglichkeiten. Als Argument bekommt dieser Parameter einen Adresszeiger auf eine Konfigurations-Struktur von Typ `ENABLE_TRACE_PARAMETERS`. Über diese können zusätzlich Dinge konfiguriert werden. In dieser Struktur finden sich unter anderem auch Parameter, die nicht offiziell dokumentiert sind. Beispiele für dokumentierte Einstellungsmöglichkeiten sind zwei Konfigurationsbits, über die es möglich ist, einzustellen, dass der ETW-Provider die SID des Benutzerkontos und ID der Terminal-Session, in dessen Kontext er ausgeführt wird, in den erweiterten Header des Event-Objektes schreibt. Erwähnenswert ist noch ein weiteres Konfigurationsbit, welches dazu führen soll, dass zu jedem erzeugten ETW-Event einer Manifest-basierten Provider-Klasse der Provider einen Stack-Trace in den erweiterten Header einfügt. Letztere Option besteht seit Windows 7 und sollte mit Bedacht eingesetzt werden. Es wird darauf hingewiesen, dass die Erstellung des Stack-Traces zusätzlich Zeit in Anspruch nimmt und somit zu einem zusätzlichen Overhead führt. Zudem ist die Größe eines Event-Objektes auf 64 KiB begrenzt. Ein großer Stack-Trace könnte dazu führen, dass die Maximalgröße des Event-Objektes überschritten wird, was dazu führt, dass das Event-Objekt verworfen wird. Zudem werden nur maximal 192 Stack-Frames des Stack-Trace protokolliert. Sollte der Stack mehr als 192 Frame-Element besitzen, wird dieser von der Basis bis zum 192. Element protokolliert und an dieser Stelle abgeschnitten [Mic9a].

Einige der zusätzlichen Daten, die im erweiterten Event-Header protokolliert werden können, werden nicht für die Provider-Klasse mittels `TraceEnableEx` und `TraceEnableEx2` konfiguriert, sondern für die ETW-Session. Um das Erzeugen dieser Zusatzinformationen einzustellen, steht die Funktion `TraceSetInformation` in der ETW-Controller-API bereit. Die Funktion ist auch in `evntrace.h` deklariert und wird entweder von der Bibliothek `SecHost.dll` ($>=$ Windows 8.1) oder von `AdvAPI32.dll` ($<$ windows 8.1) bereitgestellt [Mic9a].

Eine aktive ETW-Session kann über die Funktion `ControlTraceA/W` gesteuert werden. Dies gilt auch für fremde ETW-Sessions, die durch eine fremde Controller-Anwendung angelegt wurden. Die Funktion kann dazu entweder ein Handle auf das Session-Objekt, dass z.B. von `StartTraceA/W` geliefert wird, als Argument übergeben werden oder der

Namen einer existierenden ETW-Session als Zeichenkette. Über einen weiteren Parameter mit dem Bezeichner `ControlCode` kann spezifiziert werden, welche Aktion auf die ETW-Session angewendet werden soll. Im Folgenden sind die Aktionen, die auf eine aktive ETW-Session angewendet werden können, aufgelistet:

`STOP`	stoppt die ETW-Session
`UPDATE`	aktualisiert die Session-Konfiguration (Konfigurationsparameter können als Argument (Zeiger auf Config-Struktur) der Funktion ControlTraceA/W übergeben werden)
`FLUSH`	fordert den Logger-Thread der Session auf, den Inhalt aller Puffer der Session unmittelbar an eine Consumer-Anwendung zuzustellen oder in die ETL-Datei zu schreiben
`QUERY`	liefert statistische Informationen über den Zustand der Session und des Logging-Prozesses (Füllstand, Event-Menge, Anzahl verlorener Event-Objekte usw.)
`INCREMENT_FILE`	wechselt die ETL-Log-Datei (seit Windows 10 20H2 Update)
`CONVERT_TO_REALTIME`	wandelt die aktive ETW-Session im laufenden Betrieb von einer Buffered-Session zu einer Real-Time-Session (seit Windows 10 20H2 Update)

Tabelle 3.11: Aktionen der Funktion `ControlTraceA/W` [Mic9a]

Alternativ können für einige häufig genutzte Funktionalitäten statt der `ControlTraceA/W`-Funktion auch spezielle Funktionen wie `UpdateTraceA/W`, um die Konfiguration der Session zu aktualisieren, `FlushTraceA/W`, um die Puffer der Session zu leeren, und `StopTraceA/W`, um die Session zu stoppen, verwendet werden.

Die Deklarationen der Funktionen `ControlTraceA/W`, `UpdatetraceA/W`, `FlushTraceA/W` und `StopTraceA/W` werden auch über die Header-Datei evntrace.h bereitgestellt. Exportiert werden die Funktionen ebenfalls von SecHost.dll ($>=$ Windows 8.1) oder AdvAPI32.dll ($<$ Windows 8.1).

3.6.10.2 Anlegen und steuern einer ETW-System-Session

Wie im Abschnitt ETW-System-Logging erklärt, können System-Sessions seit Windows 8 in etwa wie normale ETW-Sessions verwaltet werden. Es besteht jedoch die Einschränkung, dass System-Sessions generell nur System-Provider-Klassen und davon nur eine pro System-Session abonnieren können. Dazu wird zum Anlegen einer neuen ETW-System-Session die bereits beschriebene Funktion `StartTraceA/W` benutzt. Diese bekommt einen Adresszeiger auf die bereits beschriebene Konfigurations-Struktur `EVENT_TRACE_PROPERTIES` übergeben. In dieser muss der Wert des Attributes

`LogFileMode` auf `EVENT_TRACE_SYSTEM_LOGGER_MODE` eingestellt werden. Diese Einstellung ist wichtig, damit die ETW-Session eine System-Session ist. Ansonsten kann die Session im Prinzip in der gleichen Weise wie normale ETW-Sessions konfiguriert werden. Nachdem die ETW-System-Session gestartet ist, kann ein ETW-System-Provider als Event-Objekt-Quelle für die Session abonniert werden. Dazu ist die bereits beschriebene Funktion `EnableTrace` zu verwenden, die für das Abonnieren von MOF-Provider-Klassen benutzt wird, da die zwei System-Provider-Klassen ebenfalls MOF-Provider-Klassen sind. Die System-Provider-Klasse wird über ihre GUID spezifiziert. Die GUIDs der zwei möglichen System-Provider-Klassen „NT Kernel Trace“ und “Circular Kernel Context„ sind im Abschnitt „ETW-System-Logging“ angegeben. Zudem kann die Keyword-Filter-Bitmaske bezüglich der Keywords, die „NT Kernel Trace“ oder „Circular Kernel Context“ definieren, konfiguriert werden. Die Schlüsselwörter, Bitmaske und Ereignistypen der „NT Kernel Trace“-Provider-Klasse sind im Abschnitt „NT Kernel Trace“ aufgelistet. Auch bei System-Sessions können im aktiven Zustand die Einstellungen der System-Session mittels `ContrlTraceA/W` wie beschrieben aktualisiert werden. Die Filter-Bitmaske kann ebenfalls im aktiven Zustand über einen erneuten Aufruf der Funktion `EnableTrace` angepasst werden. Die Funktion `EnableTrace` ermöglicht zudem den System-Provider wieder abzubestellen. Ist ein System-Provider abonniert und wird versucht, mittels `EnableTrace` einen weiteren zu abonnieren, indem als Argument eine andere GUID verwendet wird, scheitert die Anfrage und die Funktion liefert einen Fehler-Code zurück [Mic9b].

Wie im Abschnitt „ETW-System-Logging“ erläutert, können auch alternativ zum Erstellen neuer ETW-System-Sessions eine der zwei vorhandenen ETW-System-Sessions „NT Kernel Logger“ und „Circular Kernel Context Logger“ verwendet werden. Vor Windows 8 bestand nicht die Möglichkeit als Controller eigene ETW-System-Sessions anzulegen. Anwendungen mussten in Windows-Version vor Windows 8 zwangsläufig eine der beiden vorhandenen System-Sessions nutzen, wenn sie System-Logging betreiben wollten. Um eine der vorhandenen ETW-System-Sessions zu starten, wird die bereits beschriebene Funktion `StartTraceA/W` verwendet. In der `EVENT_TRACE_PROPERTIE`-Struktur, deren Adresse der Funktion `StartTraceA` als Argument übergeben wird, ist im Feld `LoggerName` der Bezeichner der vorhandenen ETW-System-Session anzugeben. Dies sind die Zeichenketten „NT Kernel Logger“ oder „Circular Kernel Context Logger“. Sollte beim Aufruf der Funktion `StartTraceA/W` die spezifizierte Session bereits verwendet werden, scheitert der Vorgang. In diesem Fall meldet die Funktion den Fehler-Code `ERROR_ALREADY_EXISTS` zurück. Anders als bei selbst erzeugten ETW-System-Sessions kann für die zwei vorhandenen System-Sessions die spezifizierte System-Provider-Klassen nicht abonniert oder abbestellt werden. Aus dem Grund scheitert jeder Aufruf der Funktion `EnableTrace`. Die Filter-Bitmaske wird in diesen zwei vordefinierten Sessions über das Spezial-Feld `EnableFlags` in der Konfigurations-Struktur

`EVENT_TRACE_PROPERTIES` spezifiziert. Die Filter-Konfiguration für Provider-Klassen „NT Kernel Trace“ oder „Circular Kernel Context“ können somit direkt beim Starten der Session durch die Funktion `StartTraceA/W` spezifiziert werden. Im aktiven Betrieb der Session kann mittels der Funktion `ControlTraceA/W` diese Einstellung aktualisiert werden. System-Sessions können wie normale Sessions über die Funktionen `ControlTraceA/W` oder `StopTraceA/W` gestoppt werden. Dies sollte insbesondere bei den vordefinierten System-Sessions getan werden, damit diese anderen Anwendungen für das System-Logging zur Verfügung stehen [Mic9c].

3.6.10.3 Anlegen und steuern von ETW-Sessions mittels Kommandozeilen-Programmen

Unter Windows existieren Kommandozeilen-Anwendungen, welche die Rolle des ETW-Controllers einnehmen und zum Konfigurieren und Steuern von ETW-Sessions verwendet werden können. So kann eine Software, statt selbst die Controller-Funktionalität zu implementieren auch Kommandozeilenaufrufe tätigen, und diese Programme als Werkzeug benutzen. Die Kommandozeilen-Tools könnten als Kindprozess, mit entsprechenden Argumenten aufgerufen, stellvertretend die Controller-Aktionen durchführen. Auch Administratoren oder Angreifer können diese Tools über eine Kommandozeilenverbindung (Terminal-Session) benutzen [Mic9b].

Beispiele sind die Comand-Line-Tools `xperf` [Mic2b], `logman` [Mic2a] und `wevtutil` [Mic2d]. Alternativ kann die Verwaltung, Konfiguration und Steuerung von ETW-Sessions auch über spezielle ETW-Befehle der `PowerShell` durchgeführt werden [Mic2c].

3.6.11 Event-Konsumierung und -Dekodierung (Consumer)

Der Erhalt von ETW-Event-Objekten aus ETW-Sessions ist offiziell nur User-Mode-Code möglich. Zumindest stehen die dafür notwendigen APIs nur als User-Mode-Bibliotheken zur Verfügung. Kernel-Treibern ist es damit offiziell nicht möglich ETW-Events zu empfangen. Eine ETW-Consumer-Anwendung kann die Event-Objekte entweder im Real-Time-Betrieb direkt aus den Session-Puffern erhalten, wenn die ETW-Session dies erlaubt, oder diese nach einer durchgeführten Protokollierung aus einer ETL-Datei auslesen. Für beide Varianten ist offiziell die ETW-Consumer-API zu verwenden. Diese wird durch die User-Mode-Bibliothek `SecHost.dll` bereitgestellt [MicAa][AIRS21, S. 512ff].

Um Event-Objekte aus ETW-Sessions oder aus ETL-Protokollierungen zu beziehen, muss der Prozess sich als ETW-Consumer für eine bestimmte ETW-Session, aus der er direkt Event-Objekte im Real-Time-Betrieb geliefert bekommen möchte, oder für eine

ETL-Datei, die er verarbeiten möchte, beim ETW-System anmelden. Dabei kann ein Consumer-Prozess nur für eine ETW-Real-Time-Session gleichzeitig als Consumer angemeldet sein. Gleichwohl ist es möglich für mehrere ETL-Datei-Analysen parallel dazu angemeldet zu sein. Für jede Anmeldung ist die Datenstruktur `EVENT_TRACE_LOGFILE` anzulegen und entsprechend mit Werten zu füllen [Mic5a]. Der Inhalt der Struktur spezifiziert die Anmelde-Konfiguration für den Consumer beim ETW-System. In der Struktur ist entweder bei Nutzung des Real-Time-Betriebs der Name der ETW-Session als Zeichenkette im Feld `LoggerName` oder beim Beziehen der Event-Objekte aus ETL-Dateien der Dateipfad als Zeichenkette im Feld `LogFileName` zu spezifizieren. Darüber hinaus können einige weitere Dinge konfiguriert werden. Entscheidend sind die drei Callback-Routinen `EventRecordCallback`, `EventCallback` und `BufferCallback`, die über Funktionspointer in der Konfigurations-Struktur hinterlegt werden können.
Die `EventRecordCallback` und `EventCallback` werden von der ETW-Consumer-API aufgerufen, wenn im Real-Time-Betrieb ein neues ETW-Event-Objekt von der ETW-Session aus dem Kernel empfangen wurde. Bei der Verarbeitung von protokollierten Event-Objekten aus einer ETL-Datei werden die Callback-Funktionen von der User-Mode-Bibliothek im Zuge des Datei-Lese-Vorgangs mehrfach hintereinander aufgerufen, bis alle in der ETL-Datei spezifizierten Event-Objekte ausgelesen sind.
Die `EventRecordCallback`-Funktion liefert über ihre Parameter unter anderem die binär kodierten Daten des Event-Objektes als `EVENT_RECORD`-Struktur und bezieht sich auf Event-Daten von Manifest-basierten Providern. Dahingegen liefert die `EventCallback`-Funktion über ihre Parameter die binär kodierten Daten des Event-Objektes als `EVENT_TRACE`-Struktur und bezieht sich auf Event-Objekte von MOF-Providern. Die `BufferCallback`-Funktion wird durch die ETW-Consumer-API aufgerufen, wenn ein ETW-Session-Buffer voll ist oder aufgrund eines Flush-Vorgangs durch den ETW-Logger-Thread geleert wurde. Sie liefert damit regelmäßig statistische Daten über den Puffer- und Logging-Status der ETW-Session. Aus den Daten kann z.B. entnommen werden, wie viele Event-Objekte in die Session-Puffer eingetragen wurden, vom Consumer empfangen wurden oder auf den Datenträger gespeichert wurden. Auch die Menge an verlorenen Event-Objekten, die entstehen, falls die Eingangsmenge neuer Event-Daten zu groß ist und das Weiterreichen an Consumer-Prozesse oder das Logging in ETL-Datei zu langsam verläuft, kann aus den Daten erkannt werden. [MicAa]

Ist die Anmeldung des Prozesses als Consumer beim ETW-System über die `EVENT_TRACE_LOGFILE`-Struktur konfiguriert, wird der Anmeldevorgang durch den Aufruf der Funktion `OpenTraceA/W` durchgeführt, die als Argument einen Adresszeiger auf die `EVENT_TRACE_LOGFILE`-Struktur übergeben bekommt. Die `OpenTraceA/W`-Funktion wird von der User-Mode-Bibliothek `SecHost.dll` bereitgestellt und ist in der Header-Datei `evntrace.h` deklariert. Die Funktion liefert einen Handle zurück, den der Consumer-Code für das Abfragen der Event-Objekte benötigt. Durch die `CloseTrace`-Funktion der

`SecHost.dll` (`evntrace.h`) ist die Abmeldung des Consumers am ETW-System möglich [MicAa].

Der Empfang von Event-Objekten wird von der Funktion `ProcessTrace` implementiert, die ebenfalls von `SecHost.dll` exportiert wird und über eine Deklaration in `evntrace.h` Entwicklern bereitgestellt wird. Die `ProcessTrace`-Funktion kann Event-Objekte aus mehreren Quellen (einer ETW-Real-Time-Session und mehrerer verschiedenen ETL-Dateien), für die sich als Consumer angemeldet wurde, parallel empfangen. Dazu ist der Funktion ein Zeiger auf ein Array mit den vom `OpenTrace`-Aufruf erhaltenen Handeln zu übergeben. Mit einem weiteren Argument muss die Länge des Handle-Arrays spezifiziert werden. Zusätzlich kann optional über zwei weitere Parameter der `ProcessTrace`-Funktion ein Zeitfenster als Filter spezifiziert werden. Das Zeitfenster bezieht sich nur auf Event-Objekte aus ETL-Dateien. Dabei werden nur Event-Objekte geliefert, deren Zeitstempel sich innerhalb der übergebenen Grenzen befindet. Ein Aufruf der `ProcessTrace`-Funktion blockiert den Thread, der sie aufruft, im Kernel-Code des ETW-Systems, bis ein oder mehrere neue ETW-Events in der Real-Time-Session eingegangen sind und durch den ETW-Logger-Thread bereitgestellt werden oder ein oder mehrere Event-Objekte aus der ETL-Datei gelesen wurden. Für jede Serie an neu empfangenen oder gelesenen Event-Objekten wacht der Thread auf und ruft für jedes Event-Objekt die beschriebenen Callback-Funktionen auf, in denen der Consumer-Code spezifiziert ist, der diese verarbeitet. Sind alle neu eingegangenen Event-Objekte verarbeitet, wird der Thread nach der letzten Rückkehr aus der Callback-Funktion wieder blockiert [MicAa][AIRS21, S. 511ff].

Der Zugriff auf das sich im Kernel befindende ETW-System wird durch einen Systemcall umgesetzt. Dieser wird durch die Funktion `NtTraceControl` der `NTDLL.dll` durchgeführt. `SecHost.dll` nutzt diese Funktion, um mit dem ETW-System im Kernel zu kommunizieren. Die Funktion `NtTraceControl` und der dahinter liegende Systemcall wird somit sowohl für die Anmeldung eines Consumer-Prozesses am ETW-System als auch für den Empfang der Event-Objekte aus einer ETW-Session benutzt. Dies ist eine weitere Funktionalität dieser Prozedur neben der Anmeldung und Steuerung von ETW-Providern sowie der Steuerung von ETW-Sessions durch einen Controller-Prozess. Die `NtTraceControl`-Funktion und der Systemcall besitzen einen `FunctionCode`-Parameter und zwei Adresszeiger auf allgemeine Puffer für eingehende und ausgehende Daten. Der Wert des `FunctionCode`-Parameters spezifiziert die Aktion oder Anfrage an das ETW-System. Je nach Anfrage werden die Argumente in den Eingabe-Puffer gelegt, aus dem das ETW-System im Kernel die Argumente liest. Die Kernel-Komponente schreibt die Rückgabe-Daten in den Ausgabe-Puffer. Somit können über eine Funktion und einen Systemcall verschiedenste Funktionalitäten implementiert werden [Cha20b].

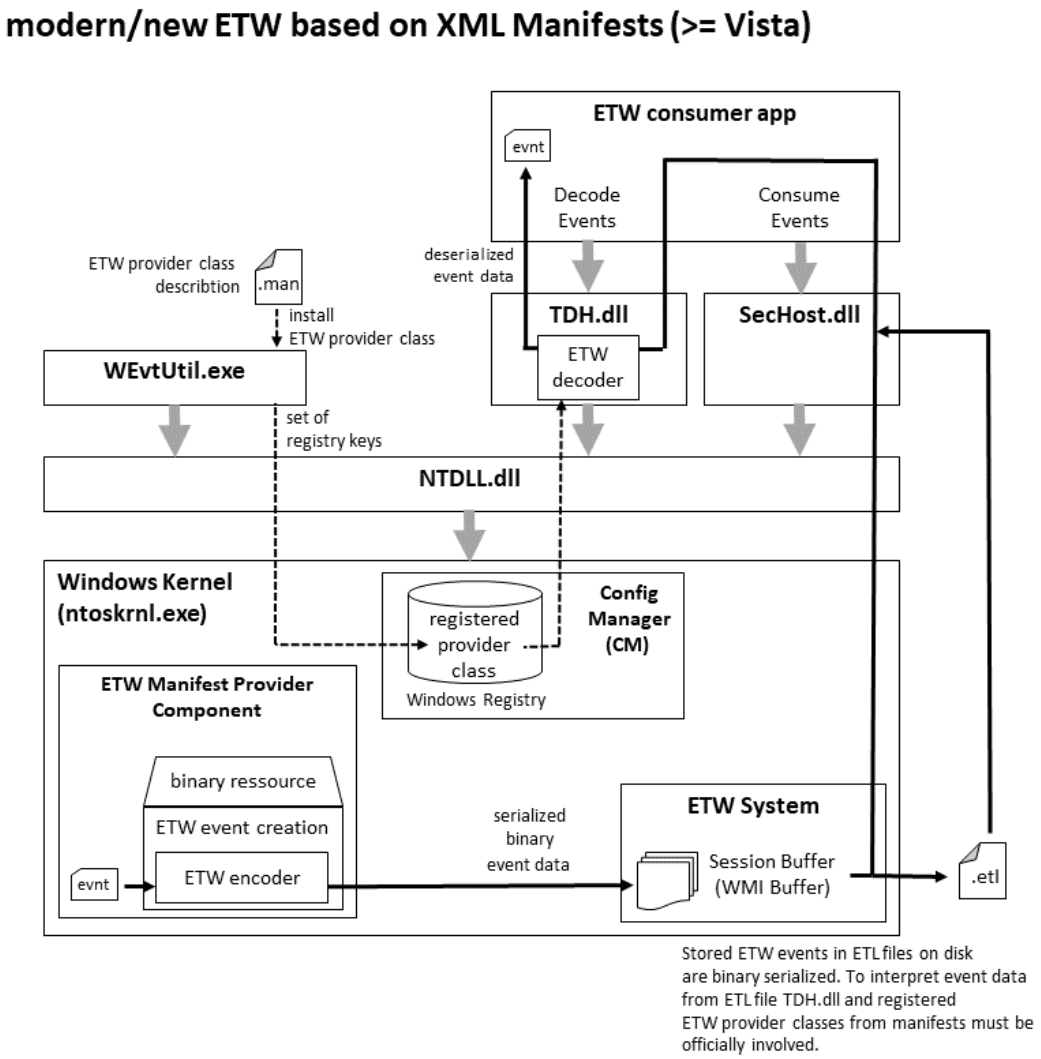

Abbildung 3.6: Konsumierung und -Dekodierung Manifest-basierter Events

Die empfangenen Event-Objekte liegen als binäre Datenstrukturen (`EVENT_RECORD`-Struktur oder `EVENT_TRACE`-Struktur) vor [Mic5c]. Aus ihnen kann der allgemeine Event-Header extrahiert und gelesen werden. Dieser gibt Aufschluss über die Provider-Klasse, den Ereignistyp sowie weiteren Eigenschaften, wie Auftrittszeitpunkt, Ausführungskontext (Prozess, Thread, CPU), Level, Op-Code und assoziierte Schlüsselwörter (Ereignistyp-spezifische Bitmaske) [Cha20c]. Die Ereignistyp-spezifischen Attribute des Event-Objektes sind jedoch noch serialisiert und kodiert. Sie befinden sich als binär kodierte Bytefolge am Ende der Datenstrukturen. Um diese als Consumer-App zu interpretieren, müssen sie erst dekodiert werden und als Datenstruktur mit Feldern, auf die zugegriffen werden kann, bereitgestellt werden. Da die Serialisierungsformate für MOF-Event-Daten und Manifest-basierte Event-Daten nicht öffentlich dokumentiert sind, ist offiziell nicht vorgesehen, diese kodierten Daten manuell zu dekodieren.

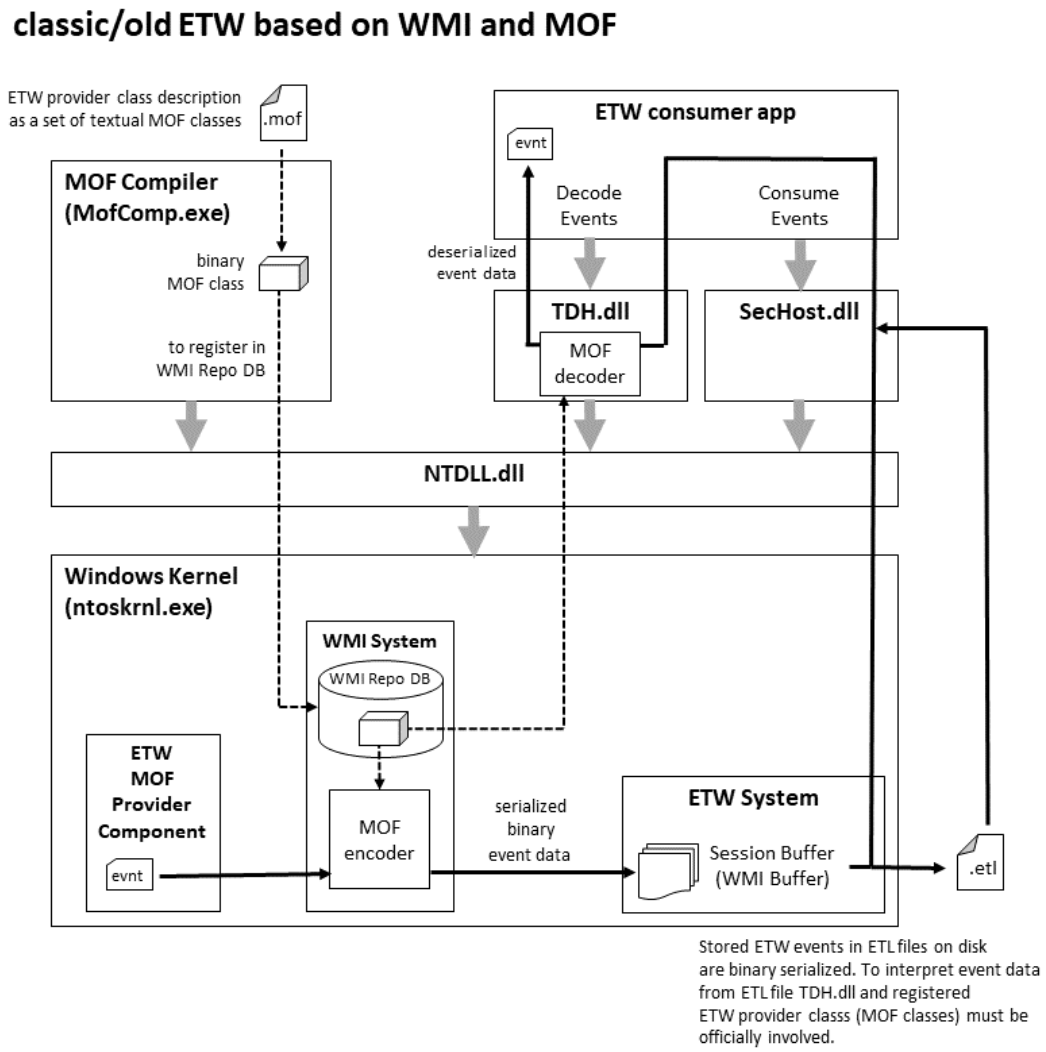

Abbildung 3.7: Konsumierung und -Dekodierung MOF-basierter Events

Stattdessen bietet Microsoft mit der User-Mode-Bibliothek `TDH.dll` (Trace Decoder Helper) Funktionen an, mittels derer die Nutzdaten mit den Attribut-Werten der Event-Objekte dekodiert und aufbereitet als C-Strukturen dem Consumer-Code bereitgestellt werden können [MicAb][AIRS21, S. 513ff]. Zu diesem Zweck bietet die `TDH.dll` die Funktion `TdhGetEventInformation` an, deren Deklaration über die Header-Datei `tdh.h` bereitgestellt wird. Die `TdhGetEventInformation`-Funktion dekodiert die Daten des gesamten empfangenen Event-Records und füllt eine `TRACE_EVENT_INFO`-Datenstruktur, die die Daten des ETW-Event-Objektes zugreifbar macht. Die Attribute werden als Array vom Element-Typ `EVENT_PROPERTY_INFO` bereitgestellt. Jedes Array-Element liefert den Wert eines Attributes des Event-Objektes. Die Struktur `EVENT_PROPERTY_INFO` liefert neben dem Wert des Attributes auch eine Meta-Beschreibung des Attributs mit Datentyp und Bezeichner als Zeichenketten. Für die Dekodierung der Event-Daten benötigt die Decoder-Logik innerhalb der `TDH.dll` die Strukturbeschreibungen der Event-Objekte für die verschiedenen Ereignistypen, die dekodiert werden sollen. Diese sind Teil

der installierten Provider-Klassen. Je nach Klassen-Typ ruft der Code in `TDH.dll` die benötigten Informationen ab [MicAb]. Für das Dekodieren von Event-Objekten Manifest-basierter Provider-Klassen werden die Struktur-Informationen über die Ereignistypen aus der Windows Registry abgerufen. Installierte Manifest-Provider-Klassen sind als Registry-Schlüssel unter `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\WINEVT\` in der Registrierungsdatenbank abgelegt [AIRS21, S. 506ff]. Bei installierten MOF-Provider-Klassen sind die Struktur-Informationen in Form von kompilierten MOF-Klassen in der WMI-Repository-Datenbank (WMI-Repo-DB) des WMI-Systems abgelegt. Der Code in `TDH.dll` greift im Falle einer Dekodierung eines Event-Objektes, das auf einer MOF-Provider-Klasse basiert, selbstständig auf das WMI-System zu und lädt die für das Dekodieren benötigten MOF-Beschreibungen [AIRS21, S. 506ff].

3.6.12 Startprotokollierung mittels ETW-Autologger

Event-Tracing-for-Windows (ETW) bietet die Möglichkeit, bereits während des Bootvorgangs und in der frühen Initialisierungsphase Events in einer Session zu sammeln und diese in eine ETL-Datei zu protokollieren. Möglich wird dies durch eine Funktion, die als „ETW-Autologger“ bezeichnet wird [Mic9d][AIRS21, S. 521ff]. Das ETW-System wird sehr früh im Initialisierungsprozess des Windows-Kernels eingerichtet, bevor die meisten anderen Basis-Funktionalitäten des Windows-Kernels initialisiert werden. Dies ermöglicht diesen Komponenten bereits bei ihrer Initialisierung Ereignisse dem ETW-System zu melden, sollte eine entsprechende ETW-Session bestehen. Direkt nachdem das ETW-System sich selbst initialisiert hat, startet es die konfigurierten ETW-Autologger-Sessions [AIRS21, S. 501ff][Mic9d]. Autologger-Sessions können benutzerseitig vor einem Neustart konfiguriert werden. Die Konfiguration wird in der Registry persistent gespeichert, sodass sie nach dem Neustart verfügbar ist. Beim Neustart entnimmt das ETW-Autologger-System die Konfiguration für die Autologger-Sessions aus der Registry des Config-Managers, richtet die ETW-Sessions ein und startet diese automatisch. Der Config-Manager und die Kernel-internen Funktionen zum Lesen der Hive-Files vom Laufwerk sind bereits vor dem ETW-System initialisiert [AIRS21, S. 824ff][AIRS21, S. 501ff]. Durch die sehr früh im Initialisierungsprozess laufenden Autologger-Sessions ist es möglich, ab diesem Zeitpunkt Event-Objekte von verschiedenen Providern zu sammeln und in ETL-Dateien zu speichern.

Die ETW-API bietet keine Funktionen, um Autologger-Sessions vor einem Neustart zu konfigurieren. Stattdessen soll die Konfiguration laut Microsoft-Dokumentation benutzerseitig über manuell zu erstellende Registry-Einträge geschehen. Natürlich könnten diese auch von einer4 Controller-Anwendung erzeugt werden. Unter dem Schlüssel `HKLM\SYSTEM\CurrentControlSet\Control\WMI\Autologger` ist für jede Autologger-Session ein Schlüssel anzulegen mit frei definierbarem Namen. Der Schlüssel repräsentiert

eine Autologger-Session. Für den Schlüssel muss ein Satz von Werte-Feldern definiert werden, welche die Session-Konfiguration bilden. Für jede zu abonnierende Provider-Klasse ist ein Unter-Schlüssel anzulegen, der als Bezeichner die GUID der zu abonnierenden Provider-Klasse besitzt. Auch für jeden dieser Schlüssel sind eine Reihe von Werte-Feldern zu hinterlegen, die das Abonnement der Provider-Klasse näher spezifizieren. Über diese Felder können z.B. unter anderem die Filter-Keywords spezifiziert werden [Mic9d].

Listing 3.3: Registry-Konfiguration zur Erstellung einer ETW-Autologger-Session (Quelle: Microsoft API-Dokumentation [Mic9d])

```
HKEY_LOCAL_MACHINE (HKLM)
  \SYSTEM
     \CurrentControlSet
        \Control
           \WMI
              \Autologger
                 \< LoggerSessionA >
                    \< ProviderGuid1 >
                    \< ProviderGuid2 >
                 \< LoggerSessionB >
                 \< LoggerSessionC >
```

Die ETW-Autologger-Funktionalität existiert seit Windows Vista und löst die Vorgänger-Methode „ETW-Global-Logger“ ab. Die Global-Logger-Funktionalität existiert aus Gründen der Abwärtskompatibilität weiterhin. Ab Windows Vista wird jedoch empfohlen, anstelle von Global-Logger den neueren ETW-Autologger zu benutzen [Mic9d]. Aus diesem Grund, ist die Global-Logger-Funktionalität in dieser Arbeit nicht näher beschrieben.

3.6.13 DC- und Rundown-Ereignisse

Manche ETW-Provider-Klassen definieren Ereignistypen, welche die Zeichenketten „`DC`“ und „`Rundown`“ im Namen tragen. So liefert die ETW-System-Provider-Klasse „NT Kernel Trace“ mehrere Ereignistypen, welche den Bezeichner „`DCStart`“ und „`DCStop`“ tragen. In manchen Manifest-basierten Provider-Klassen sind Ereignistypen mit den Worten „`Rundown`“, „`StartUpRundown`“, „`StartRundown`“, „`EndRundown`“ oder „`StopRundown`“ in ihren Bezeichnern zu finden. Diese Ereignisse werden in der Regel von den für die Provider-Klasse angemeldeten Provider-Prozessen erstellt, wenn diese über die Callback-Funktion benachrichtigt werden, dass sie aktiviert oder deaktiviert wurden und zusätzlich über die Keyword-Filtermaske spezifiziert wurde, dass Rundown-Events erzeugt werden sollen. Diese Rundown-Ereignisse entstehen also direkt nach dem Abonnieren einer ETW-Provider-Klasse oder direkt vor dem Abbestellen dieser durch die angebundenen Provider-Prozesse. Auch das Starten oder Stoppen einer ETW-Session kann dazu führen,

dass ETW-Provider von abonnierten Klassen Rundown-Events senden. Beim Deaktivieren eines Providers, erstellt und sendet dieser die Events, wenn die Aufforderung zum Deaktivieren eingeht, noch bevor er seinen Zustand auf deaktiviert wechselt. Besonders relevant können oft die Startup-Rundown-Ereignisse sein, die von den Providern direkt bei ihrer Aktivierung gesendet werden. Über die `DCStart`-Ereignisse teilen die Kernel-Provider der System-Provider-Klasse NT Kernel Trace“ z.B. mit, welche Prozesse und Threads bereits bei ihrer Aktivierung existieren und welche Module und Bibliotheken von den verschiedenen Prozessen bereits geladen wurden. Dies ermöglicht Analyseanwendungen als ETW-Consumer eine Datenbasis aufzubauen, für die Ereignis-Daten von bereits vergangenen Vorgängen notwendig wären. So kann eine Monitoring-Anwendung z.B. durch die Startup-Rundown-Ereignisse alle Informationen erhalten, die es zum Bau eines Prozessbaums und einer vollständigen Thread-Tabelle benötigt, obwohl die Erstellung der Prozesse und Threads vor dem Start der Analyseanwendung und der ETW-Session stattgefunden haben. Zwar gäbe es auch alternative Wege für eine Anwendung diese Daten zu beschaffen, jedoch bieten manche ETW-Provider-Klassen und die dahinter liegenden Provider Consumer-Anwendungen durch Rundown-Ereignisse einen sehr komfortablen Weg, diese Daten direkt durch des ETW-System in Form von ETW-Events zu beschaffen [Sol20].

3.7 NT Kernel Trace

In diesem Abschnitt sind die Ereignistypen und Schlüsselwörter der ETW-System-Provider-Klasse „NT Kernel Trace“ aufgelistet und beschrieben. Die System-Provider-Klasse „NT Kernel Trace“ wird seit Windows 10 auch als „SystemTraceProvider“ bezeichnet.

3.7.1 Alle Ereignistypen des NT-Kernel-Trace

Name	Bit	Wert	Konstante	Keyword
Prozesse, Threads und Jobs (Erzeugung und Beendigung)				
PROCESS	0	0x 00 00 00 01	EVENT_TRACE_FLAG_PROCESS	process
THREAD	1	0x 00 00 00 02	EVENT_TRACE_FLAG_THREAD	thread
JOB	19	0x 00 08 00 00	EVENT_TRACE_FLAG_JOB	job
Images (Laden und Entladen)				
IMAGE LOAD	2	0x 00 00 00 04	EVENT_TRACE_FLAG_IMAGE_LOAD	img
Systemcalls				
SYSTEMCALL	7	0x 00 00 00 80	EVENT_TRACE_FLAG_SYSTEMCALL	syscall

Tabelle 3.12: „NT Kernel Trace“-Quellen (geordnet) (Teil 1) [Mic5f]

Name	Bit	Wert	Konstante	Keyword
Ausnahmebehandlung und Kontextwechsel				
INTERRUPT	6	0x 00 00 00 40	EVENT_TRACE_FLAG_INTERRUPT	isr
DPC	5	0x 00 00 00 20	EVENT_TRACE_FLAG_DPC	dpc
DISPATCHER	11	0x 00 00 08 00	EVENT_TRACE_FLAG_DISPATCHER	dispatcher
CONTEXT SWITCH	4	0x 00 00 00 10	EVENT_TRACE_FLAG_CSWITCH	cswitch
Memory Management (Paging)				
PAGE FAULT	12	0x 00 00 10 00	EVENT_TRACE_FLAG_MEMORY_PAGE_FAULT	pf
HARD PAGE FAULT	13	0x 00 00 20 00	EVENT_TRACE_FLAG_MEMORY_HARD_FAULT	hf
VIRTUAL ALLOCATION	14	0x 00 00 40 00	EVENT_TRACE_FLAG_VIRTUAL_ALLOC	virtalloc
VRITUAL ADDRESS MAPPING	15	0x 00 00 80 00	EVENT_TRACE_FLAG_VAMAP	vamap
Datei-Zugriffe (Datei-System)				
FILE IO INIT	26	0x 04 00 00 00	EVENT_TRACE_FLAG_FILE_IO_INIT	fileio
FILE IO COMPLETION	25	0x 02 00 00 00	EVENT_TRACE_FLAG_FILE_IO	fileiocomplet
FILE IO DISK MAPPING	9	0x 00 00 02 00	EVENT_TRACE_FLAG_DISK_FILE_IO	file
Datenträger-Zugriffe (Storage-Driver/Disk-IO)				
DISK IO INIT	10	0x 00 00 04 00	EVENT_TRACE_FLAG_DISK_IO_INIT	diskinit
DISK IO COMPLETION	8	0x 00 00 01 00	EVENT_TRACE_FLAG_DISK_IO	disk
Registry-Zugriffe (Config-Manager)				
REGISTRY	17	0x 00 02 00 00	EVENT_TRACE_FLAG_REGISTRY	registry
Interprozesskommunikation (IPC)				
ALPC	20	0x 00 10 00 00	EVENT_TRACE_FLAG_ALPC	alpc
Netzwerk-Stapel (TCP, UDP, IP usw.)				
NETWORK STACK	16	0x 00 01 00 00	EVENT_TRACE_FLAG_NETWORK_TCPIP	net
Zugriffe auf Geräte und Treiber (IO-Manager)				
DRIVER	23	0x 00 80 00 00	EVENT_TRACE_FLAG_DRIVER	driver
SPLIT IO	21	0x 00 20 00 00	EVENT_TRACE_FLAG_SPLIT_IO	splitio
Performance-Counter				
PROCESS COUNTER	3	0x 00 00 00 08	EVENT_TRACE_FLAG_PROCESS_COUNTER	proccntr
Debugging und Profiling				
DEBUG PRINT	18	0x 00 04 00 00	EVENT_TRACE_FLAG_DBGPRINT	dbgprint
DEBUG	22	0x 00 40 00 00	(undokumentiert)	–
PROFILE	24	0x 01 00 00 00	EVENT_TRACE_FLAG_PROFILE	profile

Tabelle 3.13: „NT Kernel Trace“-Quellen (geordnet) (Teil 2) [Mic5f]

Name	Bit	Wert	Konstante	Keyword
PROCESS	0	0x 00 00 00 01	EVENT_TRACE_FLAG_PROCESS	process
THREAD	1	0x 00 00 00 02	EVENT_TRACE_FLAG_THREAD	thread
IMAGE LOAD	2	0x 00 00 00 04	EVENT_TRACE_FLAG_IMAGE_LOAD	img
PROCESS COUNTER	3	0x 00 00 00 08	EVENT_TRACE_FLAG_PROCESS_COUNTER	proccntr
CONTEXT SWITCH	4	0x 00 00 00 10	EVENT_TRACE_FLAG_CSWITCH	cswitch
DPC	5	0x 00 00 00 20	EVENT_TRACE_FLAG_DPC	dpc
INTERRUPT	6	0x 00 00 00 40	EVENT_TRACE_FLAG_INTERRUPT	isr
SYSTEMCALL	7	0x 00 00 00 80	EVENT_TRACE_FLAG_SYSTEMCALL	syscall
DISK IO COMPLETION	8	0x 00 00 01 00	EVENT_TRACE_FLAG_DISK_IO	disk
FILE IO DISK MAPPING	9	0x 00 00 02 00	EVENT_TRACE_FLAG_DISK_FILE_IO	file
DISK IO INIT	10	0x 00 00 04 00	EVENT_TRACE_FLAG_DISK_IO_INIT	diskinit
DISPATCHER	11	0x 00 00 08 00	EVENT_TRACE_FLAG_DISPATCHER	dispatcher
PAGE FAULT	12	0x 00 00 10 00	EVENT_TRACE_FLAG_MEMORY_PAGE_FAULT	pf
HARD PAGE FAULT	13	0x 00 00 20 00	EVENT_TRACE_FLAG_MEMORY_HARD_FAULT	hf
VIRTUAL ALLOCATION	14	0x 00 00 40 00	EVENT_TRACE_FLAG_VIRTUAL_ALLOC	virtalloc
VRITUAL ADDRESS MAPPING	15	0x 00 00 80 00	EVENT_TRACE_FLAG_VAMAP	vamap
NETWORK STACK	16	0x 00 01 00 00	EVENT_TRACE_FLAG_NETWORK_TCPIP	net
REGISTRY	17	0x 00 02 00 00	EVENT_TRACE_FLAG_REGISTRY	registry
DEBUG PRINT	18	0x 00 04 00 00	EVENT_TRACE_FLAG_DBGPRINT	dbgprint
JOB	19	0x 00 08 00 00	EVENT_TRACE_FLAG_JOB	job
ALPC	20	0x 00 10 00 00	EVENT_TRACE_FLAG_ALPC	alpc
SPLIT IO	21	0x 00 20 00 00	EVENT_TRACE_FLAG_SPLIT_IO	splitio
DEBUG	22	0x 00 40 00 00	(undokumentiert) [Cha20a]	–
DRIVER	23	0x 00 80 00 00	EVENT_TRACE_FLAG_DRIVER	driver
PROFILE	24	0x 01 00 00 00	EVENT_TRACE_FLAG_PROFILE	profile
FILE IO COMPLETION	25	0x 02 00 00 00	EVENT_TRACE_FLAG_FILE_IO	fileiocomplet
FILE IO INIT	26	0x 04 00 00 00	EVENT_TRACE_FLAG_FILE_IO_INIT	fileio
–	27	0x 08 00 00 00	(undokumentiert)	–
< no config >	28	0x 10 00 00 00	EVENT_TRACE_FLAG_NO_SYSCONFIG	–
< enable reserve >	29	0x 20 00 00 00	(undokumentiert) [Cha20a]	–
< wmi forwarding >	30	0x 40 00 00 00	(undokumentiert [Cha20a]	–
< extended Flags >	31	0x 80 00 00 00	(undokumentiert) [Cha20a]	–

Tabelle 3.14: „NT Kernel Trace“-Quellen (32Bit-Filter-Bitmaske) [Cha20a][Mic5f]

3.7.1.1 Prozesse, Threads und Jobs (Erzeugung und Beendigung)

PROCESS

Bit-Nummer:	0	Beschreibung:
Wert:	0x 00 00 00 01	Protokolliert das Erzeugen und Beenden
Bezeichner:	EVENT_TRACE_FLAG_PROCESS	von Prozessen.

ETW-Quelle: Process
Event-Format: Process_TypeGroup1

Events	
1	Start
2	End
3	DCStart
4	DCEnd
39	Defunc

Parameter	
uint32	UniqueProcessKey
uint32	ProcessId
uint32	ParentId
uint32	SessionId
sint32	ExitStatus
uint32	DirectoryTableBase
object	UserSID
string	ImageFileName
string	CommandLine

Tabelle 3.15: NT Kernel Trace - PROCESS [Mic5f]

THREAD

Bit-Nummer:	1	Beschreibung:
Wert:	0x 00 00 00 02	Protokolliert das Starten und Stoppen
Bezeichner:	EVENT_TRACE_FLAG_THREAD	von Threads.

ETW-Quelle:	Thread
Event-Format:	Thread_TypeGroup1

Events	
1	Start
2	End
3	DCStart
4	DCEnd

Parameter	
uint32	ProcessId
uint32	ThreadId
uint32	StackBase
uint32	StackLimit
uint32	UserStackBase
uint32	UserStackLimit
uint32	Affinity
uint32	Win32StartAddr
uint32	TebBase
uint32	SubProcessTag
uint8	BasePriority
uint8	PagePriority
uint8	IoPriority
uint8	ThreadFlags

Tabelle 3.16: NT Kernel Trace - THREAD [Mic5f]

JOB

Bit-Nummer:	19	Beschreibung:
Wert:	0x 00 08 00 00	Protokolliert Aktivitäten im Kontext von
Bezeichner:	EVENT_TRACE_FLAG_JOB	Jobs (undokumentiert) (>= Win 10).

ETW-Quelle:	NA
Event-Format:	NA

Events	Parameter
NA	NA

Tabelle 3.17: NT Kernel Trace - JOB [Mic5f][Cha20a]

3.7.1.2 Images (Laden und Entladen)

IMAGE LOAD

Bit-Nummer:	2	Beschreibung:
Wert:	`0x 00 00 00 04`	Protokolliert das Laden und Entladen
Bezeichner:	`EVENT_TRACE_FLAG_IMAGE_LOAD`	nativer Bibliotheken (ImageLoad).

ETW-Quelle: `Image`
Event-Format: `Image_Load`

Events	
10	Load
2	Unload
3	DCStart
4	DCEnd

Parameter	
uint32	ImageBase
uint32	ImageSize
uint32	ProcessId
uint32	ImageCheckSum
uint32	TimeDateStamp
uint32	Reserved0
uint32	DefaultBase
uint32	Reserved1
uint32	Reserved2
uint32	Reserved3
uint32	Reserved4
string	FileName

Tabelle 3.18: NT Kernel Trace - IMAGE LOAD [Mic5f]

3.7.1.3 Systemcalls

SYSTEMCALL

Bit-Nummer:	7	Beschreibung:
Wert:	0x 00 00 00 80	Protokolliert das Einspringen in und das
Bezeichner:	EVENT_TRACE_FLAG_SYSTEMCALL	Rückkehren aus Systemroutinen (>= Vista).

ETW-Quelle:	PerfInfo
Event-Format:	SysCallEnter

Events	Parameter
51 SysCIEnter	uint32 SysCallAddress

ETW-Quelle:	PerfInfo
Event-Format:	SysCallExit

Events	Parameter
52 SysCIExit	uint32 SysCallNtStatus

Tabelle 3.19: NT Kernel Trace - SYSTEMCALL [Mic5f]

3.7.1.4 Ausnahmebehandlung und Kontextwechsel

INTERRUPT

Bit-Nummer:	6	Beschreibung:
Wert:	0x 00 00 00 40	Protokolliert das Auftreten von Interrupts
Bezeichner:	EVENT_TRACE_FLAG_INTERRUPT	(>= Vista).

ETW-Quelle:	PerfInfo
Event-Format:	ISR

Events	Parameter
67 ISR	object InitialTime uint32 Routine uint8 ReturnValue uint8 Vector uint16 Reserved

Tabelle 3.20: NT Kernel Trace - INTERRUPT [Mic5f]

DPC

Bit-Nummer:	5	Beschreibung:
Wert:	0x 00 00 00 20	Protokolliert das Auftreten von DPCs
Bezeichner:	EVENT_TRACE_FLAG_DPC	(>= Vista).

ETW-Quelle: PerfInfo
Event-Format: DPC

Events	
66	ThreadDPC
68	DPC
69	TimerDPC

Parameter	
object	InitialTime
uint32	Routine

Tabelle 3.21: NT Kernel Trace - DPC [Mic5f]

DISPATCHER

Bit-Nummer:	11	Beschreibung:
Wert:	0x 00 00 08 00	Protokolliert den Dispatch von Threads
Bezeichner:	EVENT_TRACE_FLAG_DISPATCHER	(ThreadReady) (>= Vista).

ETW-Quelle: Thread
Event-Format: ReadyThread

Events	
50	ReadyThread

Parameter	
uint32	TThreadId
sint8	AdjustReason
sint8	AdjustIncrement
sint8	Flag
sint8	Reserved

Tabelle 3.22: NT Kernel Trace - DISPATCHER [Mic5f]

CONTEXT SWITCH

Bit-Nummer:	4	Beschreibung:
Wert:	0x 00 00 00 10	Protokolliert Wechsel des
Bezeichner:	EVENT_TRACE_FLAG_CSWITCH	Thread-Kontextes (>= Vista).

ETW-Quelle:	Thread
Event-Format:	CSwitch

Events	
36	CSwitch

Parameter	
uint32	NewThreadId
uint32	OldThreadId
sint8	NewThreadPriority
sint8	PreviousCState
sint8	SpareByte
sint8	OldThreadWaitReason
sint8	OldThreadWaitMode
sint8	OldThreadState
sint8	OldThreadWaitIdealProcessor
uint32	NewThreadWaitTime
uint32	Reserved

Tabelle 3.23: NT Kernel Trace - CONTEXT SWITCH [Mic5f]

3.7.1.5 Memory Management (Paging)

PAGE FAULT

Bit-Nummer:	12	Beschreibung:
Wert:	0x 00 00 10 00	Protokolliert alle Arten von Page-Faults.
Bezeichner:	EVENT_TRACE_FLAG_MEMORY_PAGE_FAULT	

ETW-Quelle:	PageFault_V2
Event-Format:	PageFault_TypeGroup1

Events	
10	TransitionFault
11	DemandZeroFault
12	CopyOnWrite
13	GuardPageFault
14	HardPageFault
15	AccessViolation

Parameter	
object	InitialTime
uint64	ReadOffset
uint32	VirtualAddress
uint32	FileObject
uint32	TThreadId
uint32	ByteCount

Tabelle 3.24: NT Kernel Trace - PAGE FAULT [Mic5f]

HARD PAGE FAULT

Bit-Nummer:	13	Beschreibung:
Wert:	0x 00 00 20 00	Protokolliert Hard-Page-Faults.
Bezeichner:	EVENT_TRACE_FLAG_MEMORY_HARD_FAULT	

ETW-Quelle:	PageFault_V2
Event-Format:	PageFault_HardFault

Events	
32	HardFault

Parameter	
object	InitialTime
uint64	ReadOffset
uint32	VirtualAddress
uint32	FileObject
uint32	TThreadId
uint32	ByteCount

Tabelle 3.25: NT Kernel Trace - HARD PAGE FAULT [Mic5f]

VIRTUAL ALLOCATION

Bit-Nummer:	14	Beschreibung:
Wert:	0x 00 00 20 00	Protokolliert Aufrufe von VirtualAlloc und
Bezeichner:	EVENT_TRACE_FLAG_VIRTUAL_ALLOC	VirtualFree (>= Vista).

ETW-Quelle:	PageFault_V2
Event-Format:	PageFault_VirtualAlloc

Events	
98	VirtualAlloc
99	VirtualFree

Parameter	
uint32	BaseAddress
object	RegionSize
uint32	ProcessId
uint32	Flags

Tabelle 3.26: NT Kernel Trace - VIRTUAL ALLOCATION [Mic5f]

VIRTUAL ADDRESS MAPPING

Bit-Nummer:	14	Beschreibung:
Wert:	0x 00 00 20 00	Protokolliert Zuordnungsvorgänge in
Bezeichner:	EVENT_TRACE_FLAG_VAMAP	virtuellen Adressräumen (>= Win 8).

ETW-Quelle:	NA
Event-Format:	NA

Events	Parameter
NA (map, unmap ???) [Cha20a]	NA

Tabelle 3.27: NT Kernel Trace - VIRTUAL ALLOCATION [Mic5f][Cha20a]

3.7.1.6 Datei-Zugriffe (Datei-System)

FILE IO INIT

Bit-Nummer:	26	Beschreibung:
Wert:	0x 04 00 00 00	Protokolliert den Beginn von
Bezeichner:	EVENT_TRACE_FLAG_FILE_IO_INIT	Datei-Operationen (>= Vista).

ETW-Quelle: FileIo
Event-Format: FileIo_Create

Events	
64	Create

Parameter	
uint32	IrpPtr
uint32	TTID
uint32	FileObject
uint32	CreateOptions
uint32	FileAttributes
uint32	ShareAccess
string	OpenPath

ETW-Quelle: FileIo
Event-Format: FileIo_DirEnum

Events	
72	DirEnum
77	DirNotify

Parameter	
uint32	IrpPtr
uint32	TTID
uint32	FileObject
uint32	FileKey
uint32	Length
uint32	InfoClass
uint32	FileIndex
string	FileName

ETW-Quelle: FileIo
Event-Format: FileIo_Info

Events	
69	SetInfo
70	Delete
71	Rename
74	QueryInfo
75	FSControl

Parameter	
uint32	IrpPtr
uint32	TTID
uint32	FileObject
uint32	FileKey
uint32	ExtraInfo
uint32	InfoClass

Tabelle 3.28: NT Kernel Trace - FILE IO INIT (Teil 1) [Mic5f]

ETW-Quelle:	`FileIo`
Event-Format:	`FileIo_ReadWrite`

Events	
67	Read
68	Write

Parameter	
uint64	Offset
uint32	IrpPtr
uint32	TTID
uint32	FileObject
uint32	FileKey
uint32	IoSize
uint32	IoFlags

ETW-Quelle:	`FileIo`
Event-Format:	`FileIo_SimpleOp`

Events	
65	Cleanup
66	Close
73	Flush

Parameter	
uint32	IrpPtr
uint32	TTID
uint32	FileObject
uint32	FileKey

Tabelle 3.29: NT Kernel Trace - FILE IO INIT (Teil 2) [Mic5f]

FILE IO

Bit-Nummer:	25	Beschreibung:	
Wert:	`0x 02 00 00 00`	Protokolliert das Fertigstellen von	
Bezeichner:	`EVENT_TRACE_FLAG_FILE_IO`	Datei-Operationen (>= Vista).	

ETW-Quelle:	`FileIo`
Event-Format:	`FileIo_OpEnd`

Events	
76	OperationEnd

Parameter	
uint32	IrpPtr
uint32	ExtraInfo
uint32	NtStatus

Tabelle 3.30: NT Kernel Trace - FILE IO [Mic5f]

DISK FILE IO

Bit-Nummer:	9	Beschreibung:
Wert:	0x 00 00 02 00	Protokolliert die Zuordnung zwischen
Bezeichner:	EVENT_TRACE_FLAG_DISK_FILE_IO	File-IDs und File-Names (Kernel Objects).

ETW-Quelle: FileIo
Event-Format: FileIo_Name

Events		Parameter	
0	Name	uint32	FileObject
32	FileCreate	string	FileName
35	FileDelete		
36	FileRundown		

Tabelle 3.31: NT Kernel Trace - DISK FILE IO [Mic5f]

3.7.1.7 Datenträger-Zugriffe (Storage-Driver/Disk-IO)

DISK IO INIT

Bit-Nummer:	10	Beschreibung:
Wert:	0x 00 00 04 00	Protokolliert den Beginn eines Zugriffs
Bezeichner:	EVENT_TRACE_FLAG_DISK_IO_INIT	auf ein Massenspeichergerät (>= Vista).

ETW-Quelle: DiskIo
Event-Format: DiskIo_TypeGroup2

Events		Parameter	
12	ReadInit	uint32	Irp
13	WriteInit	uint32	IssuingThreadId
15	FlushInit		

Tabelle 3.32: NT Kernel Trace - DISK IO INIT [Mic5f]

DISK IO

Bit-Nummer:	8	Beschreibung:
Wert:	0x 00 00 01 00	Protokolliert das Fertigstellen eines Zugriffs
Bezeichner:	EVENT_TRACE_FLAG_DISK_IO	auf ein Massenspeichergerät.

ETW-Quelle: DiskIo
Event-Format: DiskIo_TypeGroup1

Events

10	Read
11	Write

Parameter

uint32	DiskNumber
uint32	IrpFlags
uint32	TransferSize
uint32	Reserved
sint64	ByteOffset
uint32	FileObject
uint32	Irp
uint64	HighResResponseTime
uint32	IssuingThreadId

ETW-Quelle: DiskIo
Event-Format: DiskIo_TypeGroup3

Events

14	FlushBuffers

Parameter

uint32	DiskNumber
uint32	IrpFlags
uint64	HighResResponseTime
sint32	Irp
uint32	IssuingThreadId

Tabelle 3.33: NT Kernel Trace - DISK IO [Mic5f]

3.7.1.8 Registry-Zugriffe (Config-Manager)

REGISTRY

Bit-Nummer:	17	Beschreibung:
Wert:	0x 00 02 00 00	Protokolliert Zugriffe auf die
Bezeichner:	EVENT_TRACE_FLAG_REGISTRY	Windows Registry (Config-Manager).

ETW-Quelle: Registry
Event-Format: Registry_TypeGroup1

Events

10	Create
11	Open
12	Delete
13	Query
14	SetValue
15	DeleteValue
16	QueryValue
17	EnumerateKey
18	EnumerateValueKey
19	QueryMultipleValue
20	SetInformation
21	Flush
22	KCBCreate
23	KCBDelete
24	KCBRundownBegin
25	KCBRundownEnd
26	Virtualize
27	Close

Parameter

sint64	InitialTime
uint32	Status
uint32	Index
uint32	KeyHandle
string	KeyName

Tabelle 3.34: NT Kernel Trace - REGISTRY [Mic5f]

3.7.1.9 Interprozesskommunikation (IPC)

ALPC

Bit-Nummer:	20	Beschreibung:
Wert:	0x 00 10 00 00	Protokolliert das Auftreten von ALPCs.
Bezeichner:	EVENT_TRACE_FLAG_ALPC	

ETW-Quelle:	ALPC
Event-Format:	ALPC_Send_Message

Events	Parameter
33 ALPC-Send-Message	uint32 MessageID

ETW-Quelle:	ALPC
Event-Format:	ALPC_Receive_Message

Events	Parameter
34 ALPC-Receive-Message	uint32 MessageID

ETW-Quelle:	ALPC
Event-Format:	ALPC_Wait_For_Reply

Events	Parameter
35 ALPC-Wait-For-Reply	uint32 MessageID

ETW-Quelle:	ALPC
Event-Format:	ALPC_Wait_For_New_Message

Events	Parameter
36 ALPC-Wait-For-New-Message	uint32 IsServerPort string PortName

ETW-Quelle:	ALPC
Event-Format:	ALPC_Unwait

Events	Parameter
37 ALPC-Unwait	uint32 Status

Tabelle 3.35: NT Kernel Trace - ALPC [Mic5f]

3.7.1.10 Netzwerk-Stapel (TCP, UDP, IP usw.)

NETWORK STACK

Bit-Nummer:	16	Beschreibung:
Wert:	0x 00 01 00 00	Protokolliert verschiedene Aktivitäten des
Bezeichner:	EVENT_TRACE_FLAG_NETWORK_TCPIP	Kernel-internen Network-Stack.

ETW-Quelle: TcpIp
Event-Format: TcpIp_TypeGroup1

Events	
11	RecvIPV4
13	DisconnectIPV4
14	RetransmitIPV4
16	ReconnectIPV4
18	TCPCopyIPV4

Parameter	
uint32	PID
uint32	size
object	daddr
object	saddr
object	dport
object	sport
uint32	seqnum
uint32	connid

ETW-Quelle: TcpIp
Event-Format: TcpIp_TypeGroup2

Events	
12	ConnectIPV4
15	AcceptIPV4

Parameter	
uint32	PID
uint32	size
object	daddr
object	saddr
object	dport
object	sport
uint16	mss
uint16	sackopt
uint16	tsopt
uint16	wsopt
uint32	rcvwin
sint16	rcvwinscale
sint16	sndwinscale
uint32	seqnum
uint32	connid

Tabelle 3.36: NT Kernel Trace - NETWORK STACK (Teil 1) [Mic5f]

ETW-Quelle: `TcpIp`
Event-Format: `TcpIp_TypeGroup3`

Events	
27	RevcIPV6
29	DisconnectIPV6
30	RetransmitIPV6
32	REconnectIPV6
34	TCPCopyIPV6

Parameter	
uint32	PID
uint32	size
object	daddr
object	saddr
object	dport
object	sport
uint32	seqnum
uint32	connid

ETW-Quelle: `TcpIp`
Event-Format: `TcpIp_TypeGroup4`

Events	
28	ConnectIPV6
31	AcceptIPV6

Parameter	
uint32	PID
uint32	size
object	daddr
object	saddr
object	dport
object	sport
uint16	mss
uint16	sackopt
uint16	tsopt
uint16	wsopt
uint32	rcvwin
sint16	rcvwinscale
sint16	sndwinscale
uint32	seqnum
uint32	connid

Tabelle 3.37: NT Kernel Trace - NETWORK STACK (Teil 2) [Mic5f]

ETW-Quelle: `TcpIp`
Event-Format: `TcpIp_SendIPV4`

Events	Parameter
10 SendIPV4	uint32 PID uint32 size object daddr object saddr object dport object sport uint32 startime uint32 endtime uint32 seqnum uint32 connid

ETW-Quelle: `TcpIp`
Event-Format: `TcpIp_SendIPV6`

Events	Parameter
26 SendIPV6	uint32 PID uint32 size object daddr object saddr object dport object sport uint32 startime uint32 endtime uint32 seqnum uint32 connid

ETW-Quelle: `TcpIp`
Event-Format: `TcpIp_Fail`

Events	Parameter
17 Fail	uint16 Proto uint16 FailureCode

Tabelle 3.38: NT Kernel Trace - NETWORK STACK (Teil 3) [Mic5f]

ETW-Quelle: `UdpIp`
Event-Format: `UdpIp_TypeGroup1`

Events	
10	SendIPV4
11	RecvIPV4

Parameter	
uint32	PID
uint32	size
object	daddr
object	saddr
object	dport
object	sport
uint32	seqnum
uint32	connid

ETW-Quelle: `UdpIp`
Event-Format: `UdpIp_TypeGroup2`

Events	
26	SendIPV6
27	RecvIPV6

Parameter	
uint32	PID
uint32	size
object	daddr
object	saddr
object	dport
object	sport
uint32	seqnum
uint32	connid

ETW-Quelle: `UdpIp`
Event-Format: `UdpIp_Fail`

Events	
17	Fail

Parameter	
uint16	Proto
uint16	FailureCode

Tabelle 3.39: NT Kernel Trace - NETWORK STACK (Teil 4) [Mic5f]

3.7.1.11 Zugriffe auf Geräte und Treiber (IO-Manager)

DRIVER

Bit-Nummer:	23	Beschreibung:
Wert:	0x 00 80 00 00	Protokolliert Anfragen an Gerätetreiber
Bezeichner:	EVENT_TRACE_FLAG_DRIVER	(IO-Manager) (>= Vista).

ETW-Quelle: DiskIo
Event-Format: DriverCompleteRequest

Events	Parameter
52 DrvComplReq	uint32 RoutineAddr uint32 Irp uint32 UniqMatchId

ETW-Quelle: DiskIo
Event-Format: DriverCompleteRequestReturn

Events	Parameter
53 DrvComplReqRet	uint32 Irp uint32 UniqMatchId

ETW-Quelle: DiskIo
Event-Format: DriverCompletionRoutine

Events	Parameter
37 DrvComplRout	uint32 Routine uint32 Irp uint32 UniqMatchId

ETW-Quelle: DiskIo
Event-Format: DriverMajorFunctionCall

Events	Parameter
34 DrvMjFnCall	uint32 MajorFunction uint32 MinorFunction uint32 RoutineAddr uint32 FileObject uint32 Irp uint32 UniqMatchId

ETW-Quelle: DiskIo
Event-Format: DriverMajorFunctionReturn

Events	Parameter
35 DrvMjFnRet	uint32 Irp uint32 UniqMatchId

Tabelle 3.40: NT Kernel Trace - DRIVER [Mic5f]

SPLIT IO

Bit-Nummer:	21	Beschreibung:
Wert:	0x 00 20 00 00	Protokolliert IO-Vorgänge, die gesplittet
Bezeichner:	EVENT_TRACE_FLAG_SPLIT_IO	wurden (z.B. aufgrund von Mirroring).

ETW-Quelle:	SplitIo
Event-Format:	SplitIo_Info

Events		Parameter	
32	SplitIO	uint32	ParentIrp
		uint32	ChildIrp

Tabelle 3.41: NT Kernel Trace - SPLIT IO [Mic5f]

3.7.1.12 Performance-Counter

PROCESS COUNTERS

Bit-Nummer:	3	Beschreibung:
Wert:	0x 00 00 00 08	Liefert „Windows Performance Counter“-
Bezeichner:	EVENT_TRACE_FLAG_PROCESS_COUNTERS	Werte als ETW-Ereignisse (>= Vista).

ETW-Quelle:	Process_V2
Event-Format:	Process_V2_TypeGroup2

Events		Parameter	
32	PerfCtr	uint32	ProcessId
33	PerfCtrRundown	uint32	PageFaultCount
		uint32	HandleCount
		uint32	Reserved
		object	PeakVirtualSize
		object	PeakWorkingSetSize
		object	PeakPagefileUsage
		object	QuotaPeakPagedPoolUsage
		object	QuotaPeakNonPagedPoolUsage
		object	VirtualSize
		object	WorkingSetSize
		object	PagefileUsage
		object	QuotaPagedPoolUsage
		object	QuotaNonPagedPoolUsage
		object	PrivatePageCount

Tabelle 3.42: NT Kernel Trace - PROCESS COUNTERS [Mic5f]

3.7.1.13 Debugging und Profiling

DEBUG PRINT

Bit-Nummer:	18	Beschreibung:
Wert:	0x 00 04 00 00	Liefert Debug-Ausgaben (DbgPrint,
Bezeichner:	EVENT_TRACE_FLAG_DBGPRINT	DbgPrintEx) als ETW-Ereignisse.

ETW-Quelle:	NA
Event-Format:	NA

Events	Parameter
NA (DbgPrint, DbgPrintEx ???) [Cha20a]	NA

Tabelle 3.43: NT Kernel Trace - DEBUG PRINT [Mic5f][Cha20a]

DEBUG

Bit-Nummer:	22	Beschreibung:
Wert:	0x 00 40 00 00	(nicht dokumentiert)
Bezeichner:	EVENT_TRACE_FLAG_DEBUG	

ETW-Quelle:	NA
Event-Format:	NA

Events	Parameter
NA	NA

Tabelle 3.44: NT Kernel Trace - DEBUG [Mic5f][Cha20a]

PROFILE

Bit-Nummer:	24	Beschreibung:
Wert:	0x 01 00 00 00	Liefert jede Millisekunde Ereignisse für ein
Bezeichner:	EVENT_TRACE_FLAG_PROFILE	„Sampled Based Profiling“ (>= Vista)

ETW-Quelle:	PerfInfo
Event-Format:	SampledProfile

Events	Parameter
46 SampleProfile	uint32 InstructionPointer uint32 ThreadId uint32 Count

Tabelle 3.45: NT Kernel Trace - PROFILE [Mic5f]

3.7.2 Für die Angriffserkennung interessante Ereignistypen des NT-Kernel-Trace

Prozesse und Threads (Erzeugung und Beendigung)

PROCESS	0	0x 00 00 00 01	EVENT_TRACE_FLAG_PROCESS	process
THREAD	1	0x 00 00 00 02	EVENT_TRACE_FLAG_THREAD	thread

Erkennung von Prozesserzeugungen durch die Malware oder einen entführter Prozess (Infektionskette)
Erkennung der Verwendung von Kommandozeilentools durch einen bösartigen Prozess
Erkennung des Startvorgangs bösartiger Threads

Images (Laden und Entladen)

IMAGE LOAD	2	0x 00 00 00 04	EVENT_TRACE_FLAG_IMAGE_LOAD	img

Erkennung des Ladens bösartiger Bibliotheken (Infektionskette)
Erkennung des Nachladens von Bibliotheken durch einen Malware-Prozess (Funktionserweiterung)

Datei-Zugriffe (Datei-System)

FILE IO INIT	26	0x 04 00 00 00	EVENT_TRACE_FLAG_FILE_IO_INIT	fileio

Erkennung von Dateizugriffen durch die Malware
(Ablegen von Malware, Stehlen von Daten, Verschlüsseln, Beschädigen, Löschen)
Erkennung von IPC-Nutzung (Mutex, Pipes usw.)

Datenträger-Zugriffe (Storage-Driver/Disk-IO)

DISK IO INIT	10	0x 00 00 04 00	EVENT_TRACE_FLAG_DISK_IO_INIT	diskinit

Erkennung von Direkt-Zugriffen auf die Daten eines Datenträgers am Dateisystem vorbei

Registry-Zugriffe (Config-Manager)

REGISTRY	17	0x 00 02 00 00	EVENT_TRACE_FLAG_REGISTRY	registry

Erkennung von Änderungen der Systemkonfiguration durch einen bösartigen Prozess
(Persistierung, Störung/Abschaltung von Detektions- und Abwehrmaßnahmen)
Erkennung des Auslesens bestimmter Systemkonfigurationen durch einen bösartigen Prozess
(Stehlen von Informationen, Erkennung des Zielsystems)

Interprozesskommunikation (IPC)

ALPC	20	0x 00 10 00 00	EVENT_TRACE_FLAG_ALPC	alpc

Erkennung von Kommunikation zwischen Malware-Prozessen
Erkennung der Nutzung von Diensten und der Kommunikation mit den Dienst-Prozessen

Netzwerk-Stapel (TCP, UDP, IP usw.)

NETWORK STACK	16	0x 00 01 00 00	EVENT_TRACE_FLAG_NETWORK_TCPIP	net

Erkennung von Kommunikation mit Command-Control-Systemen (c2)
Erkennung von Angriffen auf andere Rechner im Netzwerk (lateral movement)
Erkennung von Spreading-Verhalten über das Netzwerk/Internet

Zugriffe auf Geräte und Treiber (IO-Manager)

DRIVER	23	0x 00 80 00 00	EVENT_TRACE_FLAG_DRIVER	driver

Erkennung von Zugriffen auf Geräte bzw. Treiber-Funktionen
Eventuell Erkennung von Angriffen auf Treiber-Module

Tabelle 3.46: Für die Angriffserkennung interessante Quellen des NT-Kernel-Trace

PROCESS (für Angriffserkennung relevante Ereignisse)

Eventname	Beschreibung	Parameter (nur relevante)
ETW-Quelle: Process		
DCStart	bereits ausgeführter Prozess	ProcessId, ParentId, ImageFileName, CommandLine, SessionID, UserSID
Start	Prozessstart	ProcessId, ParentId, ImageFileName, CommandLine, SessionID, UserSID
Stop	Prozessende	ProcessId, ParentId, ImageFileName, CommandLine, SessionID, UserSID

Tabelle 3.47: Für die Angriffserkennung relevante „PROCESS“-Ereignisse

THREAD (für Angriffserkennung relevante Ereignisse)

Eventname	Beschreibung	Parameter (nur relevante)
ETW-Quelle: Thread		
DCStart	bereits ausgeführter Thread	ThreadId, ProcessId, Win32StartAddr
Start	Threadstart	ThreadId, ProcessId, Win32StartAddr
Stop	Threadende	ThreadId, ProcessId, Win32StartAddr

Tabelle 3.48: Für die Angriffserkennung relevante „THREAD“-Ereignisse

IMAGE LOAD (für Angriffserkennung relevante Ereignisse)

Eventname	Beschreibung	Parameter (nur relevante)
ETW-Quelle: Image		
DCStart	bereits eingebundenes Image	FileName, ProcessId, ImageBase
Load	Image eingebunden	FileName, ProcessId, ImageBase
Unload	Image ausgebunden	FileName, ProcessId, ImageBase

Tabelle 3.49: Für die Angriffserkennung relevante „IMAGE LOAD“-Ereignisse

FILE IO INIT (für Angriffserkennung relevante Ereignisse)

Eventname	Beschreibung	Parameter (nur relevante)
ETW-Quelle: FileIo		
Create	File-Handle erstellen (Datei öffnen / Datei anlegen)	FileObject, OpenPath, CreateOptions, ShareAccess
Close	File-Handle schließen	FileObject
Read	Daten aus Datei lesen	FileObject, Offset, IOSize
Write	Daten in Datei schreiben	FileObject, Offset, IOSize
Delete	Datei löschen	FileObject
Rename	Datei umbenennen	FileObject
QueryInfo	Dateiinformationen abfragen	FileObject
SetInfo	Dateiinformationen bearbeiten	FileObject
FSControl	FSCTL-Befehl auf Datei angewendet	FileObject
DirEnum	Verzeichnis abgefragt	FileObject, FileName

Tabelle 3.50: Für die Angriffserkennung relevante „FILE IO INIT“-Ereignisse

DISK IO INIT (für Angriffserkennung relevante Ereignisse)

Eventname	Beschreibung	Parameter (nur relevante)
ETW-Quelle: DiskIo		
ReadInit	lesender Zugriff beginnen	Irp, IssuingThreadId
WriteInit	schreibender Zugriff beginnen	Irp, IssuingThreadId

Tabelle 3.51: Für die Angriffserkennung relevante „DISK IO INIT“-Ereignisse

REGISTRY (für Angriffserkennung relevante Ereignisse)

Eventname	Beschreibung	Parameter (nur relevante)
ETW-Quelle: Registry		
Create	Schlüssel erstellt	KeyHandle, KeyName, InitialTime
Open	Schlüssel geöffnet	KeyHandle, KeyName, InitialTime
Close	Schlüssel geschlossen	KeyHandle, KeyName, InitialTime
Query	Schlüssel ausgelesen	KeyHandle, KeyName, InitialTime
Delete	Schlüssel gelöscht	KeyHandle, KeyName, InitialTime
EnumerateKey	Kind-Schlüssel abgerufen	KeyHandle, KeyName, InitialTime
EnumerateValueKey	Werte abgerufen	KeyHandle, KeyName, InitialTime
QueryValue	Wert aus Schlüssel ausgelesen	KeyHandle, KeyName, InitialTime
QueryMultipleValue	mehrwertigen Wert ausgelesen	KeyHandle, KeyName, InitialTime
SetValue	Wert erstellt oder geändert	KeyHandle, KeyName, InitialTime
DeleteValue	Wert gelöscht	KeyHandle, KeyName, InitialTime

Tabelle 3.52: Für die Angriffserkennung relevante „REGISTRY“-Ereignisse

ALPC (für Angriffserkennung relevante Ereignisse)

Eventname	Beschreibung	Parameter (nur relevante)
ETW-Quelle: ALPC		
ALPC-Send-Message	ALPC-Message versendet	MessageID
ALPC-Receive-Message	ALPC-Message empfangen	MessageID

Tabelle 3.53: Für die Angriffserkennung relevante „ALPC“-Ereignisse

NETWORK STACK (für Angriffserkennung relevante Ereignisse)

Eventname	Beschreibung	Parameter (nur relevante)
ETW-Quelle: TcpIp		
ConnectIPV4	TCP-Verbindungsaufbau	connid, saddr, sport, daddr, dport
ConnectIPV6	TCP-Verbindungsaufbau	connid, saddr, sport, daddr, dport
DisconnectIPV4	TCP-Verbindungsabbau	connid, saddr, sport, daddr, dport
DisconnectIPV6	TCP-Verbindungsabbau	connid, saddr, sport, daddr, dport
SendIPV4	TCP-Daten gesendet	connid, saddr, sport, daddr, dport, size, seqNum
SendIPV6	TCP-Daten gesendet	connid, saddr, sport, daddr, dport, size, seqNum
RecvIPV4	TCP-Daten empfangen	connid, saddr, sport, daddr, dport, size, seqNum
RecvIPV6	TCP-Daten empfangen	connid, saddr, sport, daddr, dport, size, seqNum
ETW-Quelle: UdpIp		
SendIPV4	UDP-Datagramm gesendet (IPv4)	PID, daddr, dport, saddr, sport, size
SendIPV6	UDP-Datagramm gesendet (IPv6)	PID, daddr, dport, saddr, sport, size
RecvIPV4	UDP-Datagramm empfangen (IPv4)	PID, daddr, dport, saddr, sport, size
RecvIPV6	UDP-Datagramm empfangen (IPv4)	PID, daddr, dport, saddr, sport, size

Tabelle 3.54: Für die Angriffserkennung relevante „NETWORK STACK“-Ereignisse

DRIVER (für Angriffserkennung relevante Ereignisse)

Eventname	Beschreibung	Parameter (nur relevante)
ETW-Quelle: DiskIO		
DrvMjFnCall	Aufruf einer Gerätefunktion	MajorFunction, MinorFunction, RoutineAddr, FileObject, Irp, UniqMatchId

Tabelle 3.55: Für die Angriffserkennung relevante „DRIVER“-Ereignisse

3.8 Missbräuchliche Nutzung von ETW

So wie ETW auf der einen Seite als wichtiges Werkzeug in Sicherheitslösungen eingesetzt wird und somit als Sensor-Komponente von AnitiViren-Software und EDR-Endpunkt-Sensorik dafür sorgt, die Sicherheit von IT-Infrastruktur zu erhöhen, stellt Event-Tracing-for-Windows (ETW) auf der anderen Seite auch eine Gefahr dar. Denn in gleicher Weise, wie es als Komponente für die Erkennung und Abwehr von Angriffen eingesetzt wird, kann es auch von Angreifern missbräuchlich mit negativer Intention genutzt werden. Da es auf jedem Windows-System standardmäßig zur Verfügung steht und als wichtige Kernel-Komponente auch nicht deinstalliert oder deaktiviert werden kann, eignet es sich ideal für Living-off-the-Land-Angriffe (LotL) [TKG21b, min. 12:00].

Angreifern und Schadsoftware bietet ETW logischer Weise die gleiche Möglichkeit umfassend Vorgänge innerhalb des Gesamtsystems zu erkennen, zu protokollieren und die Event-Daten an eine externe Stelle auszuleiten, wie es diese Funktionalitäten Sicherheitslösungen gegenüber offeriert. ETW ermöglicht z.B. Schadsoftware eine Sniffer-Funktionalität für Datei-, Registry- und Netzwerk-Zugriffe zu implementieren. Parallel laufende Prozesse und Threads können erkannt werden und die Malware kann auf Starts und Stopps dieser reagieren [TKG21a, S. 12]. Während Schadsoftware für die Implementierung solcher Features zuvor Filter-Funktionalitäten auf Treiber-Ebene einsetzen musste und dafür gezwungen war, in den Kernel einzudringen, kann dies nun mit sehr viel geringeren Hürden über ETW aus dem User-Mode heraus realisiert werden. Die Möglichkeiten von ETW sind so weitreichend, dass sogar Keylogger-Funktionalitäten denkbar wären [TKG21b, min. 12:00]. Selbstverständlich sind für die Nutzung der entsprechenden Provider-Klassen spezielle Berechtigungen erforderlich, die ein Malware-Prozess nicht haben sollte. Schadsoftware, die es schafft ihre Berechtigungen illegal durch eine Exploitation zu erweitern (privilege escalation exploit) oder in einen fremden Prozess einzudringen, der über die notwendigen Berechtigungen verfügt, könnte sich somit jedoch in die Lage versetzen, diese Provider-Klassen in eigenen ETW-Sessions zu nutzen. Alternativ werden im nächsten Abschnitt zur Verwundbarkeit weitere Möglichkeiten aufgezeigt, dass ETW-System und deren Konfiguration anzugreifen, um dieses z.B. für eigene Interessen zu nutzen. Unter anderem bestehen Möglichkeiten, die in der Registry gespeicherten Security-Deskriptoren der ETW-Provider-Klassen mit den ACL-Tabellen zu manipulieren.

Um ETW vor der Nutzung durch Schadsoftware zu schützen, ist für manche der besonders sicherheitskritischen ETW-Provider-Klassen hinterlegt, dass diese nur in ETW-Sessions benutzt werden können, die ausschließlich von Prozessen gelesen werden können, für die die Protected-Process-Light-Eigenschaft (PPL) aktiviert ist. PPL wurde in Windows 8.1 eingeführt und ist eine zusätzliche Eigenschaft, die ein Prozess besitzen

kann. Das PPL-Level bekommt ein Prozess nur, wenn das Programm, auf dem er basiert extern wie intern durch Microsoft signiert wurde [YIRS17, S. 128ff]. Es ist jedoch Schadsoftware bekannt, die ohne eine Microsoft-Signatur in der Lage war, durch die Ausnutzung einer Sicherheitslücke einen PPL-Status zu erlangen oder in einen fremden Prozess einzudringen, der über diesen Status verfügt [TKG21a, S. 12].

Neben den genannten Sniffing-Funktionalitäten kann Malware aus bestimmten ETW-Ereignissen unter bestimmten Umständen (schlecht gehärtete Sandbox) erkennen, ob sie in einer virtualisierten Umgebung oder in einer Sandbox ausgeführt wird. Solche Umstände deuten auf eine Analyseumgebung hin. Manche Schadsoftware ändert ihr Verhalten, beendet oder löscht sich, wenn sie erkennt, dass sie in einer Analyseumgebung ausgeführt wird, um die dynamische Analyse zu erschweren [TKG21b, min. 12:00].

Ein missbräuchlicher Einsatz von ETW kann auch Schaden anrichten. So könnte ein Angreifer oder eine Malware das ETW-System und speziell die ETW-Sessions so konfigurieren, dass extrem viele Ereignisse aufgezeichnet werden. Dies würde dazu führen, dass das System stärker ausgelastet ist und eventuell nicht mehr ordnungsgemäß funktioniert. Zudem könnten auf Datenträger geschriebene Protokollierungen dazu führen, dass die Datenträger voll laufen [TKG21a, S. 12].

3.8.1 InfinityHook

Während die bisher beschriebenen Methoden eine missbräuchliche Nutzung von ETW innerhalb seiner Spezifikationen darstellt, besteht auch die Möglichkeit ETW missbräuchlich für Angriffe zu nutzen und dabei ETW außerhalb der Spezifikationen zu verwenden. Ein sehr interessantes Beispiel stellt das Rootkit-Werkzeug „InfinityHook“ dar. Die Software wird aus dem Kernel-Mode eingesetzt und setzt damit also voraus, dass der Angreifer bereits eine Möglichkeit gefunden hat, Schadcode im Kernel-Mode auszuführen [EMY19].

Eine wichtige Funktionalität von Kernel-Schadsoftware (Rootkits) ist es Systemfunktionalitäten zu instrumentieren. Dafür werden elementare Systemfunktionen gehookt. Besondere Bedeutung kommt dabei den Systemcalls zu. Durch das Umleiten der System-Funktionen über den schadhaften Code, wird es dem Rootkit möglich, Vorgänge im System zu erkennen und zu manipulieren. Dies ermöglicht z.B., die Ergebnisse von Anfragen zu manipulieren, sodass bestimmte System-Ressourcen für User-Mode-Prozesse unsichtbar sind. Dies nutzt Kernel-Mode-Schadsoftware unter anderem auch um eigene Aktivitäten und Spuren zu verstecken. Das bei Rootkit-Autoren beliebte Hooken der Systemcall-Funktionen über die System-Service-Dispatch-Tabelle, wird durch die in Windows eingebaute Kernel-Patch-Protection (KPP) unterbunden. KPP, auch PatchGuard genannt, überprüft die Integrität des Kernel-Codes und elementarer Datenstrukturen

durch Prüfsummen-Vergleiche und reagiert auf eine Integritätsverletzung mit einem kompletten Systemabsturz (Bluescreen-of-Death) [YIRS17, S. 890ff].

InfinityHook ermöglicht es Rootkits weiterhin Systemcalls, Kontextwechsel, Page-Faults, ISRs, DPCs und viele weitere Systemfunktionen zu hooken, ohne das PatchGuard (KPP) und die Hyper-V-basierte KPP-Variante HyperGuard (VBS) [YIRS17, S. 894ff] dies erkennen können. Besonders interessant ist dabei, dass InfinityHook für die Umsetzung eine Schwachstelle im ETW-System ausnutzt, die PatchGuard und HyperGuard im beschriebenen Kontext wirkungslos machen [EMY19].

Die InfinityHook-Software benutzt aus dem Kernel die exportierte Funktion `ZwTraceControl` des Windows-Kernel (`ntoskrnl.exe`), um als ETW-Consumer eine neue ETW-Session zu starten. Die Konfiguration wird so gewählt, dass z.B. ETW-Event-Objekte in der Session gesammelt werden, die Systemcalls signalisieren. Diesen Ereignistyp stellt z.B. die System-Provider-Klasse „NT Kernel Trace“ bereit. Diese kann in einer ETW-System-Session abonniert werden. Alternativ kann auch die vordefinierte System-Session „NT Kernel Logger“ gestartet werden, die ebenfalls Event-Objekte von dieser Provider-Klasse bezieht. Ist die ETW-Session angelegt, sucht der InfinityHook-Code in den Daten des ETW-Systems nach der `WMI_LOGGER_CONTEXT`-Struktur, die diese Session beschreibt. Diese kann einmal durch eine Adresszeigerkette ausgehend von dem exportierten Symbol `ntoskrnl.exe!PspHostSiloGlobals` lokalisiert werden. Der Weg führt über die Liste aller aktiven ETW-Sessions, in der jede Session geprüft werden muss, ob es sich um die beabsichtigte Session handelt [TKG21a, S. 23][TKG21b, min. 20:21]. Alternativ kann die Struktur auch über eine Adresszeiger-Varaible gefunden werden, die sich genau hinter der Variable `EtwpDebuggerData` befindet. Die acht Byte-breite Variable `EtwpDebuggerData` enthält eine 5-Byte breite Signatur, die seit Windows 7 gleich geblieben ist und den Wert `0x2C0804380C` enthält. So könnte die Variable `EtwpDebuggerData` auch über einen Signatur-Scan gefunden werden [EMY19]. Ist die `WMI_LOGGER_CONTEXT`-Struktur der neu gestarteten ETW-Session lokalisiert, wird nun ein Angriff auf diese Konfigurationsdaten durchgeführt. Dies ist möglich, da die Verwaltungsstrukturen des ETW-Systems nicht von PatchGuard (KPP) oder HyperGuard (VBS) geschützt werden. Beim Offset `0x28` befindet sich in der Struktur das Feld `GetCpuClock` vom Typ Funktionszeiger. Der Wert des Feldes ist die Adresse der Funktion, die für die Zeiterfassung des Auftrittszeitpunktes beim Eintragen des Event-Objektes in die Session aufgerufen wird. Offiziell können hier nur die Code-Adressen der unterstützten Zeitfunktionen `PpmQueryTime` (QPC), `EtwGetSystemTime` (Systemzeit) oder `EtwGetCycleCount` (Zykluszähler) spezifiziert sein. InfinityHook trägt hier die Adresse einer eigenen Hooking-Funktion ein. Dies hat zur Folge, dass bei jedem Event-Objekt, das in die ETW-Session eingetragen wird, die eingetragene Hooking-Funktion der InfinityHook-Software aufgerufen wird. Der Aufruf geschieht synchron im Kontext des ETW-Providers, der das Event-Objekt erzeugt und an das ETW-System übermittelt hat und dieses nun in die Session eingetragen

wird. Durch den Aufrufstack ist der gesamte Vorgang der ETW-Ereignissignalisierung nachzuvollziehen. Bezogen auf die Systemcall-Ereignisse bedeutet dies, dass der System-Service-Dispatcher der ETW-Provider für Systemcall-Ereignisse ist. Dieser erzeugt das Event-Objekt beim Verarbeiten des Systemcalls und ruft Funktionen im ETW-System auf, um das Event-Objekt in die Sessions eintragen zu lassen, die es abonniert haben. Dabei wird in Folge die Funktion hinter dem Funktionspointer `GetCpuClock` aufgerufen, was dazu führt, dass der Kontrollfluss in den Code von InfinityHook verzweigt. Der Systemcall verzweigt damit synchron über den Umweg über den Code des ETW-Systems in die Hooking-Funktion in InfinityHook. Sollten in der Session noch weitere Event-Objekte anderer Quellen eingehen, muss in der Hooking-Funktion eventuell evaluiert werden, ob es sich um das entsprechende Ereignis handelt, das beabsichtigt wurde zu hooken. Dies ist möglich über die Daten des Stacks. Auf die gleiche Weise kann jegliche ETW-Event-Erzeugung in allen Provider-Komponenten gehookt werden. Dies ermöglicht Rootkits eine weitreichende Manipulation des Kontrollflusses und die Instrumentierung vieler Systemereignisse [EMY19].

InfinityHook funktioniert laut den Entwicklern ab Windows 7 bis zu den aktuellsten Versionen von Windows 10. Bisher ist kein Sicherheitsupdate bekannt, dass die Funktionsweise von InfinityHook einschränkt [EMY19].

3.9 Verwundbarkeit

Da Event-Tracing-for-Windows (ETW) seit ein paar Jahren eine der wichtigsten Sensor-Quellen für EDR-Sensoren und andere Sicherheitssoftware darstellt, wird in der IT-Sicherheitsforschung zunehmend die Verwundbarkeit dieser Technik diskutiert. Es erscheint eine logische Schlussfolgerung, dass mit dem zunehmendem Einsatz von ETW in Sicherheitslösungen Angreifer im Zuge ihrer Angriffe und durch die von ihnen verwendete Malware versuchen werden, dass ETW-System zu stören oder zu umgehen, sodass ihre Aktivitäten nicht erkannt werden. In der Sicherheitsforschung untersuchen vor allem Penetration-Tester und Red-Teamer, die das unbefugte Eindringen in Systeme aus Testzwecken durchführen, die Störbarkeit von ETW [TKG21a, S. 14][TKG21b, min. 12:56]. Um Angriffsmöglichkeiten und Schwachstellen in Microsoft's ETW-Implementierung ausfindig zu machen, wird unter anderem das ETW-System „reverse engeneered“ und nach Sicherheitslücken durchsucht. Die in den letzten Jahren rapide angestiegene Anzahl an gefundener Sicherheitslücken (CVEs) lässt darauf schließen, dass die Suche nach Sicherheitslücken (CVEs) im ETW-System stark zugenommen hat [TKG21a, S. 10][TKG21b, min. 11:04].

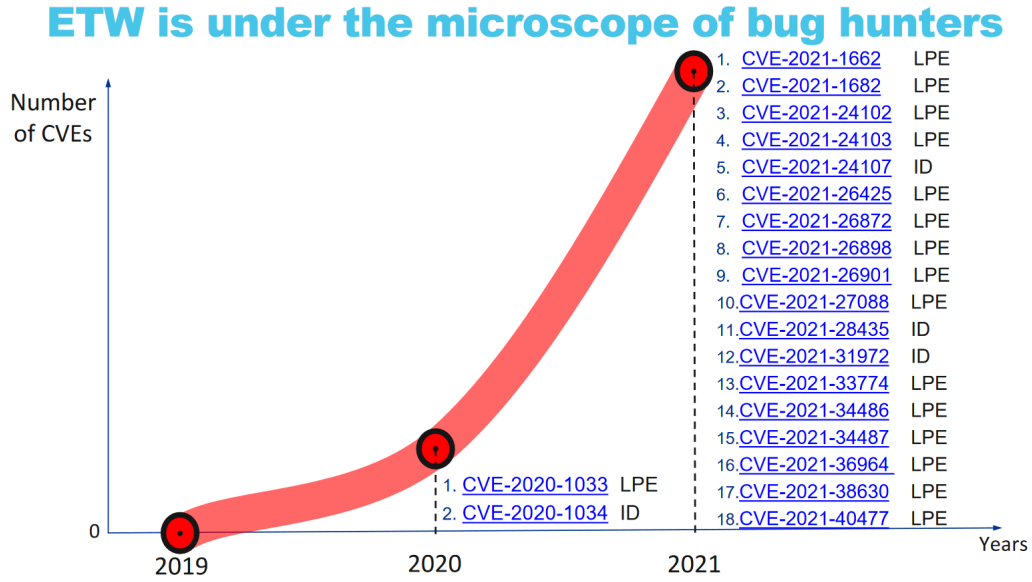

Abbildung 3.8: starker Anstieg der gefundenen Sicherheitslücken (CVEs) in ETW [TKG21b, min. 11:04]

Sicherheitsprobleme des ETW-System resultieren unter anderem auch daraus, dass ETW Closed-Source bei Microsoft entwickelt wurde und die internen Abläufe und Datenstrukturen nicht öffentlich einsehbar und dokumentiert sind. Die ETW-APIs für User-Mode-Code und Kernel-Treiber-Code bieten eine Zugriffsschicht, um ETW zu nutzen, die gleichermaßen die Interner verbirgt. Hinzu kommen etliche undokumentierte Funktionen und Datenfelder in den Strukturen, die sich zum Teil durch die API nutzen lassen. Diese existieren vermutlich deshalb, weil Microsoft ETW für die interne Entwicklung am Windows-Betriebssystem zu eigenen Zwecken nutzt und dabei für die eigene Entwicklung Möglichkeiten geschaffen wurden, die Drittparteien nicht nutzen sollen [TKG21a, S. 12]. Eventuell sollen bestimmte Funktionen nicht öffentlich zur Verfügung stehen, da sich daraus Sicherheitsprobleme ergeben könnten oder geheime Interner der Windows-Software offenbar würden. Zu den undokumentierten Dingen innerhalb der ETW-API kommen ergänzend undokumentierte Provider-Klassen und undokumentierte Ereignistypen und Attribute in öffentlichen Provider-Klassen hinzu. Auch bei letzteren ist zu vermuten, dass über diese eventuell Informationen erlangt werden könnten, die Sicherheitsprobleme implizieren. Auch durch das Vorhandensein der undokumentierten Aspekte im ETW-System wird die Wahrscheinlichkeit für bisher nicht entdeckter Sicherheitslücken erhöht. Während aufgrund fehlender Dokumentation der Sicherheitsforschung manche Schwachstellen nicht auffallen und somit nicht beseitigt werden, könnten Angreifer, die Informationen über die undokumentierten Aspekte erlangt haben, mögliche Sicherheitslücken ausnutzen [TKG21b, min. 11:35].

3.9.1 Praktischer Einsatz ETW-störender Software

Während die Möglichkeiten, Event-Tracing-for-Windows (ETW) zu stören und die Erkennung zu umgehen, in der IT-Sicherheitsforschung breit diskutiert werden und einige konkrete Software-Beispiele präsentiert wurde, die das ETW-System stört oder umgeht, wurden auf der Gegenseite reale Angriffe beobachtet, bei denen die ETW-Sensorik angegriffen wurde [TKG21a, S. 15][TKG21b, min. 13:25].

2018 wurde eine Cyber-Espionage-Malware-Plattform der APT-Gruppe „Slingshot“ durch einen von Kaspersky veröffentlichten Bericht bekannt. Die Schadsoftware benennt ETW-Log-Dateien um, um eigene in diesen Dateien aufgezeichnete Spuren zu verstecken. Die Software kam in Cyber-Angriffen gegen Ziele in Afrika und dem Mittleren-Osten zum Einsatz [Lab18][TKG21b, min. 13:25]. Die 2019 bekannt gewordene Ransomware „LockerGoga“ verursachte großen Schaden beim norwegischen Aluminiumproduzenten Norsk Hydro, deren Rechner verschlüsselt wurden, sodass der operative Betrieb ausgesetzt werden musste. LockerGoga deaktivierte ETW-Provider-Klassen für spezielle ETW-Sessions, um die in der Firma eingesetzten EDR-Sensoren zu stören [TKG21b, min. 13:25]. 2020 wurden mehrere Cyber-Angriffe im Indo-Pazifischen Raum bekannt, bei der die eingesetzte Malware-Familie ETW-Sessions deaktivierte, um EDR-Sensoren zu stören. Die Angriffe wurden der APT-Gruppe „APT 41“ zugeordnet, die aus dem chinesischen Raum heraus Espionage-Angriffe durchführt und vermutlich im Auftrag der chinesische Regierung handelt und von dieser unterstützt wird [HL21][TKG21b, min. 13:25].

Neben der in Angriffen eingesetzten Malware existieren Software-Tools der Read-Teamer und Sicherheitsforscher, die beispielhaft die Störung des ETW-Systems demonstrieren. Dazu gehören die Programme „SharpSploit“, „ScareCrow„ und „EDR-Evasion„. SharpSploit ist eine in C-Sharp (C#) geschriebene Post-Exploitation-Bibliothek zur Entwicklung von Dot-NET-Malware. Sie ist in der Lage die ETW-Sensorik für den eigenen Prozess, in dem die Bibliothek verwendet wird, zu deaktivieren [CMP21][TKG21b, min. 13:25]. ScareCrow ist ein Payload-Creation-Framework, dass es ermöglicht via Side-Loading Module in laufende Prozesse zu laden. Für diese Prozesse werden eine Reihe an EDR-Sensor-Techniken wie z.B. das API-Hooking deaktiviert. Auch die Erzeugung von ETW-Events wird für diese Prozesse so gestört, dass keine ETW-Ereignisse aus diesen Prozessen heraus signalisiert werden können. Dies unter anderem verhindern, dass der Side-Loading-Prozess sichtbar wird [OS][TKG21b, min. 13:25]. EDR-Evasion ist ein Penetration-Test-Tool, das unter anderem zehn verschiedene Verfahren zur Störung des ETW-Systems enthält[TKG21b, min. 13:25].

Die Organisation MITRE Corporation, welche die Infrastruktur einiger staatliche Forschungsinstitutionen der USA betreibt, pflegt Listen mit Methoden, die in Angriffen ein-

gesetzt werden. In der Liste für Abwehr-beeinträchtigende-Methoden (MITRE ATT&CK Impair Defense) wurde 2020 das Indicator-Blocking (ID T1562.006) aufgenommen, welches das Blockieren von Indikatoren, die von Endpunkt-Sensorik erkannt werden, beschreibt. In der Beschreibung ist die Störung des ETW-Systems ausdrücklich erwähnt [MA][TKG21a, S. 16][TKG21b, min. 15:19].

3.9.2 Angriffsfläche

In diesem Abschnitt sind konkrete Angriffsmöglichkeiten auf eine ETW-basierte Überwachungstechnik beschrieben. Die Beispiele sind dabei in acht Kategorien geordnet, die verschiedene Wege darstellen, über die eine Störung der Ereignisprotokollierung durchgeführt werden kann.

Kategorien für Angriffe auf das ETW-System:

- Angriffe über die ETW-Controller-API
- Angriffe auf die ETW-Konfiguration (Registry)
- Angriffe auf ETL-Protokollierungen (ETL-Dateien)
- Angriffe auf Provider-Konfigurationen
- Umgehen oder Stören von User-Mode-Provider-Komponenten
- Umgehen oder Stören von Kernel-Mode-Provider-Komponenten
- Angriffe auf das ETW-System im Kernel
- Angriffe über CVE-Schwachstellen
- Störung von ETW und EDR-Sensoren durch Überlastung und Fake-Ereignisse

3.9.2.1 Angriffe über die ETW-Controller-API

Ein Möglichkeit das ETW-Monitoring anzugreifen stellt das ETW-Session-Hijacking dar. Dabei versucht der Angreifer oder die Schadsoftware über die Controller-Schnittstelle fremde ETW-Sessions, die in Sicherheitslösungen genutzt werden, abzuschalten oder zu manipulieren. Das Vorgehen ist dabei in etwa immer gleich. Zuerst wird eine bestehende ETW-Session, die abgeschaltet werden soll, gestoppt. Direkt danach erstellt der Angreifer eine neue ETW-Session unter dem gleichen Namen, jedoch ohne das Abonnieren von Providern-Klassen. Dadurch bekommt die Sicherheitslösung keine Ereignisse mehr aus der Session geliefert. Erkennt die Sicherheitssoftware, dass die Session gestoppt wurde

und versucht als Controller die Session unter gleichem Namen neu zu starten, funktioniert dies nicht, da der Angreifer bereits eine Fake-Session unter diesem Namen gestartet hat [TKG21a, S. 30ff][TKG21b, min. 28:16]. Eine leisere Methode sähe so aus, dass die Session nicht komplett gestoppt wird, sondern die bestehende Session der Sicherheitslösung manipuliert wird. Beispielsweise könnten abonnierte Provider-Klassen abbestellt werden oder die Filter-Eigenschaften (Log-Level, Keywords) für die verschiedenen Abonnements so angepasst werden, dass bestimmte Event-Objekte nicht mehr protokolliert werden [TKG21a, S. 23][TKG21b, min. 20:21].

Selbstverständlich muss das Benutzerkonto, unter dem der Angreifer agiert oder in dessen Kontext die Schadsoftware ausgeführt wird, die notwendigen Berechtigungen besitzen, um auf die ETW-Sessions der Sicherheitslösung zugreifen zu dürfen. Sind diese nicht vorhanden, muss entweder eine Ausweitung der Berechtigungen (Privilege Escalation) vorangehen, sodass der Malware-Prozess die Berechtigungen besitzt, oder die Datenstrukturen des Security-Deskriptors der ETW-Sessions müssen so manipuliert werden, dass der Angreifer auf die Session zugreifen darf. Letztere Variante benötigt entweder auch erweiterte Berechtigungen, um die Security-Deskriptoren in der Registry zu manipulieren oder ein Eindringen in den Kernel, um die Security-Deskriptoren in den Verwaltungsstrukturen der ETW-Sessions im ETW-System im Kernel zu verändern [TKG21b, min. 29:19][TKG21a, S. 33ff]. Besteht die Möglichkeit die ETW-Strukturen im Kernel zu manipulieren, kann auch die Prüfung der Berechtigungen innerhalb des ETW-Kernel-Codes so manipuliert werden, dass der Zugriff ohne Prüfung erteilt wird [TKG21a, S. 36ff][TKG21a, S. 41ff][TKG21b, min. 30:01][TKG21b, min. 31:47]. Eventuell könnte eine Rechteausweitung leichter erreicht werden, in dem in einen fremden, parallel ausgeführten ETW-Controller-Prozess eingedrungen wird, der über die erforderlichen Berechtigungen verfügt. Die Controller-Aktionen könnten dann von diesem entführten Prozess aus durchgeführt werden.

Ein Angreifer oder eine Schadsoftware könnte Kommandozeilen-Anwendungen nutzen, die als ETW-Controller agieren. Beispiele wären die Anwendungen `xperf` und `logman`. Die Befehle „`xperf -stop <SessionName>`“ oder „`logman stop <SessionName> -ets`“ stoppen die ETW-Session. Mit den Befehlen „`xperf -start <SessionName> -on <Flags>`“ oder „`logman start <SessionName> -ets`“ kann die Session unter gleichem Namen wieder gestartet werden. Auch die `PowerShell` bietet Befehle zur Steuerung der ETW-Sessions. Mittels dem PowerShell-Kommando „`PS> Stop-EtwTraceSession <SessionName>`“ kann die Session gestoppt werden und mit dem Befehl „`PS> Start-EtwTraceSession <SessionNmae>`“ kann die Session wieder unter selben Namen erneut gestartet werden. Die `PowerShell` ermöglicht zudem über den Befehl „`PS> Remove-EtwTraceProvider <SessionName> <GUID>`“ eine Provider-Klasse für eine bestimmte Session abzubestellen [TKG21a, S. 23][TKG21b, min. 20:21].

Schadsoftware kann auch als ETW-Controller agieren und direkt die ETW-Controller-API nutzen. Über die Funktionen `StopTraceA/W` oder `ControlTraceA/W` kann eine existierende, aktive Session verändert oder gestoppt werden. Mittels der Funktion `StartTraceA/W` kann eine Session unter gleichem Namen wieder gestartet werden. Besitzt die Schadsoftware ein Handle auf das Session-Objekt, dass durch `StartTraceA/W` für eine neue oder eine vorhandene Session erworben werden kann, kann sie damit die Provider-Klassen-Abonnements der ETW-Session verändern. Dies ist über die Funktionen `EnableTrace`, `EnableTraceEx` und `EnableTraceEx2` möglich. Über diese Funktionen kann die Schadsoftware komplette Provider-Klassen für die Session abbestellen oder die Filter-Regeln so anpassen, dass bestimmte ETW-Events nicht mehr protokolliert werden [TKG21a, S. 30ff][TKG21b, min. 28:16][TKG21a, S. 23][TKG21b, min. 20:21].

3.9.2.2 Angriffe auf die ETW-Konfiguration (Registry)

So wie einzelne Software-Komponenten ihre Konfiguration in der Windows-Registry persistent speichern, besitzt auch das ETW-System selbst einen Satz von Registry-Schlüsseln, welche die eigenen Konfiguration enthalten. Auch diese kann Ziel von Angriffen mit dem Zweck, auf ETW basierende Sicherheitslösungen zu stören, sein. Schafft es ein Angreifer oder die Schadsoftware die Berechtigungen zu erlangen, um auf die Registry-Einträge des ETW-Systems zuzugreifen, könnte durch Änderung von Einträgen und Werten dieser das ETW-System gestört oder Zugriff auf ETW-Ressourcen erlangt werden, die daraufhin auf anderem Wege negativ manipuliert werden könnten.

Ein Beispiel hierfür sind die Security-Deskriptoren der installierten Provider-Klassen und der ETW-Sessions. Diese spezifizieren unter anderem die Access-Control-Listen (ACLs) der verschiedenen ETW-Objekte, sodass über die Secutity-Deskriptoren der Zugriff bezüglich der verschiedenen Benutzerkonten auf ETW-Sessions und ETW-Provider-Klassen gesteuert wird. Die Descurity-Deskriptoren für ETW-Sessions und Provider-Klassen sind als Bytefolge in Registry-Werte-Feldern in der Windows-Registrierungsdatenbank persistent gespeichert. Für jede ETW-Session ist unter dem Schlüssel `HKLM\System\CurrentControlSet\Control\WMI\Security\` ein Eintrag mit dem binär serialisierten Security-Deskriptor zu finden [AIRS21, S. 522ff]. Durch das Ändern des entsprechenden Eintrages für eine spezielle ETW-Session, kann der Consumer-Prozess daran gehindert werden, Event-Objekte aus der Session zu beziehen. Dies könnte z.B. erreicht werden, indem die Zugriffsberechtigung für das Benutzerkonto, unter dem der Consumer-Prozess ausgeführt wird, für die ETW-Session entzogen. Somit könnte eine Sicherheitssoftware die ETW-Events aus dieser Session konsumiert von ihrer Sensor-Quelle abgeschnitten werden [TKG21a, S. 20][TKG21a, S. 33ff][TKG21b, min. 17:35][TKG21b, min. 29:19].

Manche Sicherheitslösungen beziehen auch ETW-Events von Ereignissen, die während des Bootvorgang und der Initialisierungsphase erzeugt werden, soweit ETW dies ermöglicht. Grund dafür ist das Bedürfnis, tiefere Persistierungen und frühe Startvorgänge einer persistierten Schadsoftware zu erkennen, die während der frühen Initialisierung des Betriebssystems geladen wird. ETW ermöglicht dies, durch die Nutzung der beschriebenen Autologger-Funktionalität. Auch die Sicherheitssoftware muss dazu wie beschrieben eine entsprechende Autologger-Konfiguration in der Registry anlegen. Unter dem Schlüssel `HKLM\SYSTEM\CurrentControlSet\Control\WMI\Autologger` besteht für jede Autologger-Session ein Schlüssel mit Feldern und Unterschlüsseln. Darüber werden die Konfigurationsparameter der Autologger-Session und der abonnierten Provider-Klassen spezifiziert [AIRS21, S. 521ff][Mic9d]. Durch entfernen der Schlüssel für die Provider-Klassen oder dem Verändern der Keyword-Bitmasken-Felder, Log-Level-Felder oder Enable-Felder innerhalb dieser Schlüssel kann der Erhalt von Events aus diesen Quellen deaktiviert oder eingeschränkt werden [TKG21a, S. 20][TKG21b, min. 17:35].

3.9.2.3 Angriffe auf ETL-Protokollierungen (ETL-Dateien)

Manche Sicherheitslösungen und Analyseumgebungen verwenden die Protokollierungen in den ETL-Dateien. Dabei wird z.B. in die ETL-Dateien protokolliert und erst am Ende einer Analysesitzung (Kontext Malwareanalyse-Sandbox) oder in einem regelmäßigen Intervall auf einem Produktivsystem (z.B. jede Nacht) das ETL-Protokoll verarbeitet und eventuell ausgeleitet. Damit ein ETW-Consumer einer Sicherheitslösung, der ETL-Dateien verarbeitet, nicht mehr auf diese zugreifen kann, könnte ein Angreifer oder Schadsoftware die persistent gespeicherten Dateien auf dem Datenträger angreifen. Beispielsweise könnten diese umbenannt, versteckt oder gelöscht werden. Auch die Datei aus einer Malware heraus zu öffnen und den Inhalt zu manipulieren (bestimmte Event-Objekte entfernen) ist vorstellbar. Das Format ist zwar nicht offiziell von Microsoft öffentlich dokumentiert, trotzdem sind solche Manipulationen durch eine Schadsoftware prinzipiell möglich. So könnten forensisch relevante Spuren im Kontext einer Angriffskampagne verwischt werden [TKG21b, min. 17:35][TKG21a, S. 20][TKG21b, min. 13:25].

Auch die ETL-Dateien sind natürlich über ihre Access-Control-Listen (ACL) vor unberechtigtem Zugriff geschützt. Ein Angreifer oder Malware-Prozess muss über die notwendigen Berechtigungen verfügen, um auf die Dateien zugreifen zu dürfen.

3.9.2.4 Angriffe auf Provider-Konfigurationen

Einige Komponenten, die auch als ETW-Provider Ereignisse an das ETW-System melden, besitzen eine persistierte Konfiguration. So besitzen z.B. User-Mode-Bibliotheken oder System-Prozesse Sätze von Registry-Schlüsseln, über die sie konfiguriert sind. Die Komponenten lesen bei ihrer Initialisierung die Schlüssel bzw. deren Werte-Felder. Die Konfiguration bestimmt das Verhalten der Komponenten im Rahmen der Einstellungsmöglichkeiten, die durch die Software vorgesehen sind. Auch die Eigenschaft, ob die Komponente als ETW-Provider Ereignisse erzeugen soll, kann für manche Komponenten über die Registry konfiguriert werden.

Ein Beispiel ist die Dot-NET-Laufzeitumgebung, die als Bibliothek Teil aller Dot-Net-Prozesse ist. Über Änderungen an den Werten des Registry-Schlüssels `HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Environment` können die ETW-Provider der Dot-Net-Runtime-Bibliothek in allen Dot-NET-Prozessen deaktiviert werden [TKG21a, S. 20][TKG21b, min. 17:35]. Alternativ führt auch eine Veränderung an den zwei Umgebungsvariablen `COMPlus_ETWFlags` und `COMPLus_ETWEnabled` dazu, dass das ETW-Logging der Dot-NET-Laufzeitumgebung eingeschränkt werden kann. Durch `COMPlus_ETWEnabled` lässt sich die Event-Erzeugung komplett abschalten, während mittels `COMPlus_ETWFlags` die zu erzeugenden Ereignistypen angepasst werden können [TKG21a, S. 20][TKG21b, min. 17:35].

Zwei weitere Beispiele betreffen einen Windows-System-Prozess und eine Windows-Bibliothek. Der Windows-Systemprozess „Services“ (`services.exe`) startet die Windows-Services als Kindprozess, darunter ist auch der Service-Host-Prozess (`svchost.exe`). Mittels einer Änderung des Registry-Wertes`TracingDisabled` des Schlüssels `HKLM\Software\Microsoft\Windows NT\CurrentVersion\Tracing\SCM\Regular` kann der ETW-Provider innerhalb des Services-Prozess abgeschaltet werden. Die Windows-Bibliothek RPC-Runtime (`rpcrt4.dll`) bietet User-Mode Anwendungen eine API für die Nutzung und Konfiguration von RPC-Kanälen (RPC: Remote Procedure Call) als Mittel zur Interprozesskommunikation (IPC). Durch eine Änderung des Registry-Wertes `ExtErrorInformation` des Schlüssels `HKLM\Software\Policies\Microsoft\Windows NT\Rpc` wird die Erzeugung von Event-Objekten innerhalb des Codes der `rpcrt4.dll` für alle Prozesse, die diese Bibliothek für RPC nutzen, abgeschaltet [TKG21a, S. 20][TKG21b, min. 17:35].

Zu erwähnen ist, dass solche Registry-Einträge von fremder Software und System-Komponenten über Access-Control-Listen (ACL) geschützt sind. Die Schadsoftware oder der Angreifer muss im Kontext eines Benutzerkontos agieren, dass über die notwendigen Berechtigungen verfügt, um auf die Registry-Schlüssel zuzugreifen.

3.9.2.5 Umgehen oder Stören von User-Mode-Provider-Komponenten

Befinden sich innerhalb des Malware-Prozesses User-Mode-Bibliotheken, die ETW-Events erzeugen und an das ETW-System weiterleiten, kann dies aus Angreifersicht ungünstig sein, da eine solche Ereignisverfolgung (Tracing) Aktivitäten der Malware eventuell einer Sicherheitssoftware sichtbar macht. Das gleiche Problem existiert auch für den Schadcode, der in einem zuvor legitimen Prozess ausgeführt wird, z.B. weil dieser legitime Prozess durch eine Exploitation übernommen wurde (Code Execution Exploit) oder weil der Schadcode in einen legitimen Prozess eingebracht wurde (Code Injection). Auch in diesem Fall wird der Schadcode in einer Umgebung ausgeführt, in der sich eventuell User-Mode-Bibliotheken befinden, die Aktivitäten via ETW-Events melden. Ein populäres Beispiel sind die ETW-Events der Manifest-Provider-Klasse „Microsoft-Windows-DNS-Client“, die DNS-Anfragen von Anwendungen signalisieren. Diese Event-Objekte entstehen durch Provider innerhalb der User-Mode-DNS-API, die durch die Bibliothek `DnsAPI.dll` Anwendungen zur Verfügung steht [TKG21a, S. 21][TKG21b, min. 18:20].

Um User-Mode-ETW-Provider oder die ETW-Infrastruktur, die der Provider-Code nutzt (z.B. die Provider-API), im eigenen Prozess zu stören, kann die Schadsoftware verschiedene Methoden anwenden. User-Mode-ETW-Provider verwenden zum Erzeugen und Weiterleiten von Event-Objekten an das ETW-System die ETW-Provider-API, welche durch Funktionen der Windows-API-Bibliothek `AdvAPI32.dll` bereitgestellt werden. Effektiv ist es daher, die ETW-Logging-Funktionalitäten dieser Bibliothek zu stören. Hier können je nach Speicherschutz-Restriktionen verschiedene Methoden des Hookings und Patchens genutzt werden (Import-Table-Hooking (IAT-Hooking), Inline-Hooking, RET-Patching, Hardware-Breakpoints). Zu hooken wären z.B. die Funktionen `AdvApi32.dll!EventWrite`, `AdvApi32.dll!EtwEventWrite` und `AdvApi32.dll!RtlInitializeResource`. Wichtig ist, dass der Kontrollfluss so manipuliert wird, dass entsprechende Ereignisse nicht protokolliert werden. Neben dem kompletten Abschalten aller Event-Erzeugungen wären auch komplexere Lösungen denkbar, bei denen nur bestimmte Event-Objekte nicht erzeugt werden oder die Attribute des Event-Objektes so manipuliert werden, dass der zu versteckende Vorgang nicht mehr als solcher zu erkennen ist. Diese Hooking- und Patching-Methodik kann auch eine Ebene tiefer bei den ETW-Funktionen der `NTDLL.dll` angesetzt werden. In diesem Fall wäre der Code der Funktion `NtTraceEvent` der `NTDLL.dll` zu manipulieren [TKG21a, S. 21][TKG21b, min. 18:20].

Statt die ETW-API zu manipulieren kann auch versucht werden, Code oder Datenstrukturen der Provider-Komponenten zu patchen. Ein solches Vorgehen ist natürlich sehr spezifisch, je nachdem, um welche Komponente es sich handelt. Ein Beispiel hierfür ist

das PowerShell-Script-Logging. Dieses kann über das Feld `m_enabled` der Datenstruktur `PSEtwLogProvider` deaktiviert werden [TKG21a, S. 21][TKG21b, min. 18:20].

Die ETW-API prüft scheinbar für manche Provider-Klassen die Vertrauenswürdigkeit der Provider-Komponenten, indem geprüft wird, ob die Aufrufe der ETW-API von bestimmten Adressbereichen ausgehen, in denen z.B. der Code signierter Bibliotheken geladen wurde. Scheinbar werden für diese Prüfung die Daten aus dem Prozess-Environment-Block (PEB) benutzt. Das Patchen bestimmter Werte in der Liste geladener Module des PEB kann zu Störungen der ETW-Event-Erzeugung führen [TKG21a, S. 21][TKG21b, min. 18:20].

Neben den genannten Funktionen, die eventuell gehookt oder gepatcht werden können, existieren auch Datenstrukturen der ETW-API (`AdvAPI32.dll`), deren Manipulation zu einer Störung der Event-Erzeugung führen kann. Der Vorteil ist dabei, dass für die Daten-Adressbereiche in der Regel immer Schreib-Berechtigungen bestehen, die für die Code-Bereiche fehlen könnten. Hier ist zwischen globalen Datenstrukturen, welche die Event-Erzeugung für alle Provider-Komponenten des Prozesses stören, und lokalen Datenstrukturen, die nur die Event-Erzeugung für eine spezielle Provider-Komponente stören, zu unterscheiden [TKG21a, S. 21][TKG21b, min. 18:20].

Das Ändern der ETW-Provider-Handles, die eine Provider-Komponente bei ihrer Anmeldung vom ETW-System erhält, führt dazu, dass die Komponente nicht mehr mit dem ETW-System kommunizieren kann und somit auch keine Event-Objekte mehr erzeugen kann [TKG21a, S. 21][TKG21b, min. 18:20]. Durch Änderung der Felder `TraceLevel` und `EnableBits` in den Provider-Registrierungs-Datenstrukturen kann das Erzeugen der Event-Objekte unterbunden werden, da der Provider-Code und der API-Code vor dem Erstellen und Versenden eines Event-Objektes diese Felder prüfen und somit sicherstellen, ob der Ereignistyp überhaupt von einer Session abonniert wurde. User-Mode-Provider-Komponenten, die auf Manifest-basierten Provider-Klassen basieren, können alternativ zur ETW-API der `AdvAPI32.dll` auch einen statisch gelinkten Code für die ETW-Event-Erzeugung verwenden, der vom Message-Compiler (`MC.exe`) auf Basis des Manifestes erzeugt wurde. In diesem Fall kann die Ereignis-Signalisierung über eine Manipulation der Felder `isEnabled` und `level` der Datenstruktur `MCGEN_TRACE_CONTEXT` reguliert und abgeschaltet werden. Zudem können die Einträge für die Callback-Funktionen, die bei der Anmeldung der ETW-API bekannt gemacht werden, überschrieben werden. Die entsprechenden Callback-Funktionspointer befindet sich im Feld `Callback` der Datenstruktur `ETW_USER_REG_ENTRY`. Zusätzlich existieren Einträge in der Liste `_TigProvider!EtwpEventCallbackList` [TKG21a, S. 21][TKG21b, min. 18:20].

Eine weitere Möglichkeit für einen Schadcode die Provider-Komponenten im eigenen Prozess zu deaktivieren, ist es, diese am ETW-System abzumelden. Dafür muss der Wert des Provider-Handels gefunden werden. Mit diesem kann durch einen Aufruf der

Funktion `EtwEventUnregister` und dem Provider-Handler als Argument die Provider-Komponente abgemeldet werden, was dazu führt, dass diese keine Ereignisse via ETW mehr signalisieren kann. Bei Manifest-Providern, die den vom Message-Compiler (`MC.exe`) generierten Code als ETW-API nutzen, kann die Abmeldung des Providers über die Funktion `McGenControlCallbackV2` durchgeführt werden [TKG21a, S. 21][TKG21b, min. 18:20].

Besteht der Bedarf, die Abmeldung der Provider-Komponente nicht über Funktionen der ETW-API durchzuführen, kann auch direkt die Funktion `NtTraceControl` der `NTDLL.dll` benutzt werden oder ein entsprechender Systemcall durchgeführt werden. Für beides ist aber dennoch die Provider-Handle-Nummer erforderlich.

Letzteres Vorgehen kann auch, ohne mögliche ETW-Provider aktiv zu stören, dafür benutzt werden, eine ETW-Sensorik in User-Mode-Bibliotheken zu umgehen. So können unter Umständen für bestimmte Zugriffe direkt die Funktionen der `NTDLL.dll` benutzt werden oder Systemcalls durchgeführt werden, anstatt die API-Funktionen der Windows-API-Bibliotheken zu benutzen. Es sollte klar sein, dass ein solches Vorgehen, außerhalb der offiziell vorgesehenen Wege, wesentlich komplizierter sein kann und Schwierigkeiten verursachen kann, da möglicherweise die Systemcall-Schnittstellen über verschiedene Windows-Versionen hinweg nicht stabil ist [TKG21a, S. 22][TKG21b, min. 19:29].

3.9.2.6 Umgehen oder Stören von Kernel-Mode-Provider-Komponenten

Im Kontext eines Malware-Prozesses oder eines übernommenen Prozesses melden nicht nur ETW-Provider in User-Mode-Bibliotheken Ereignisse. Auch die verschiedenen Komponenten des Betriebssystemkernels melden als ETW-Provider Ereignisse, wodurch Aktionen des schadhaften Prozesses sichtbar gemacht werden könnten. Aus dem Kernel-Mode erzeugen sowohl der Windows-Kernel (`ntoskrnl.exe`) selbst (Prozess-Manager, Speicher-Manager, Objekt-Manager, Config-Manager usw.), als auch etliche geladene Treiber ETW-Events. Bei letzteren handelt es sich nicht nur um Gerätetreiber von Drittanbietern sondern vor allem um Windows-eigene Basistreiber, die zum Teil Bezug zu realen IO-Geräten haben, zum Teil jedoch auch Treiber für Kernel-Dienste sind, wie z.B. der Netzwerk-Stack (`TcpIp.sys`), der HTTP-Dienst (`HTTP.sys`), ICP-Schnittstellen-Dienste (`NPFS.sys`, `CimFS.sys`, `WciFS.sys`) und der Dateisystem-Dienst (`NTFS.sys`) sind [TKG21a, S. 5]. Somit kann durch die ETW-Protokollierung in Windows-eigenen Kernel-Treibern ein Großteil von Zugriffen des schadhaften Prozesses sichtbar werden. Aus diesem Grund besteht für Angreifer auch hier ein Interesse diese Kernel-Mode-Provider und die ETW-Infrastruktur, die diese Provider im Kernel nutzen, zu stören.

Die ETW-Event-Erzeugung im Kernel zu stören, ist jedoch wesentlich schwieriger als dies im User-Mode der Fall ist. Eine Schadsoftware muss dazu zwangsläufig in den Ker-

nel eindringen. Dies kann über das Laden eines eigenen Kernel-Moduls geschehen, was jedoch aufgrund fehlender und zwingend benötigter Signaturen meist nicht möglich ist. Alternativ kann der Windows-Kernel oder ein geladener Treiber durch Ausnutzung von Schwachstellen, so exploitiert werden, dass Schadcode im Kernel-Mode zur Ausführung gebracht wird. Hier sollte erwähnt werden, dass dieses Vorgehen nicht trivial ist und eine noch nicht geschlossene Sicherheitslücke voraussetzt. Daher sind solche Vorgänge eher selten und meist in fortschrittlichen Angriffen im Kontext von APT-Kampagnen zu beobachten. Im Kernel-Mode ausgeführter Schadcode besitzt die Möglichkeit den Code und die Datenstrukturen des Kernel-Spaces uneingeschränkt zu manipulieren abgesehen von Bereichen, die durch die Kernel-Patch-Protection (KPP) geschützt sind. Geladene Kernel-Treiber und die Datenstrukturen des ETW-Systems und der ETW-Kernel-API sind generell nicht durch KPP bzw. PatchGuard geschützt [TKG21a, S. 47][TKG21b, min. 36:40][YIRS17, S. 890ff]. Da der Kernel-Space Teil jedes Prozesses ist, wirken sich manipulative Änderungen an den ETW-Strukturen von Kernel-Providern prinzipiell auf die Ereigniserzeugung in allen Prozessen aus [TKG21b, min. 19:29].

Um die Event-Erzeugung zu behindern, können die Felder `IsEnabled` (Provider-Aktivierung) und `Level` (Log-Level) in der Datenstruktur `TRACE_ENABLE_INFO` eines ETW-Kernel-Providers geändert werden. Die Struktur ist über den `EtwRegHandle`, den der Provider-Code bei der Anmeldung zurückgeliefert bekommt, zu erreichen
(`EtwRegHandle : ETW_REG_ENTRY-Pointer`
`-> GuidEntry : ETW_GUID_ENTRY-Pointer`
`-> ProviderEnableInfo : TRACE_ENABLE_INFO-Pointer`).
Anstatt dem Provider vorzutäuschen, dass er deaktiviert ist oder das Log-Level herauf gesetzt ist, sodass er weniger Ereignisse signalisiert, können auch die Adresszeiger in den Strukturen geändert werden und auf eigene schadhafte Datenstrukturen umgebogen werden, sodass eine weitreichendere Manipulation des ETW-Providers möglich wird (Provider-Hijacking). Durch verändern des `GuidEntry`-Feldes kann die Information bezüglich der Provider-Klasse geändert werden. Event-Objekte könnten so eine falsche Provider-Klassen-Identifikation in ihrem Header tragen, die nicht der Provider-Klasse entspricht, für die sich der Provider-Code angemeldet hat. Werden die Adresszeiger oder das Handle mit `Null` überschrieben, kann der Provider nicht mehr mit dem ETW-System kommunizieren und keine Ereignisse mehr melden [TKG21a, S. 22][TKG21b, min. 19:29].

Basiert der ETW-Provider in einem geladenen Treiber auf einer Manifest-basierten Provider-Klasse, nutzt er in der Regel statisch gelinkten Code, der vom Message-Compiler (`MC.exe`) auf Basis des Manifestes generiert wurde, für die Event-Erzeugung. In diesem Fall können die Felder `isEnabled` und `Tracing` der Struktur `MCGEN_TRACE_CONTEXT` geändert werden, um die Event-Erzeugung zu unterbinden [TKG21a, S. 22][TKG21b, min. 19:29].

Bringt der Angreifer den Handle, den der Provider-Code bei seiner Anmeldung vom ETW-System erhält, in seine Gewalt, kann der Schadcode damit selbst ETW-Events für die entsprechende Provider-Klasse erzeugen. Das Signalisieren von Fake-Ereignissen könnte eine weitere Angriffsmöglichkeit darstellen [TKG21a, S. 22][TKG21b, min. 19:29].

3.9.2.7 Angriffe auf das ETW-System im Kernel

Ist der Angreifer in der Lage Schadcode im Kernel-Mode auszuführen, besteht als Angriffsmöglichkeit neben der Manipulation und Störung der Kernel-Provider-Komponenten auch die negative Beeinflussung des im Kernel befindlichen ETW-Systems.

Eine Variante sieht vor, die Rolle des Controllers aus dem Kernel aus einzunehmen und über die Funktion `ZwTraceControl` mit entsprechenden Op-Codes und Parametern bestehende Sessions zu stoppen, Provider-Klassen für die Sessions abzubestellen oder Parameter wie Keywords-Bitmasken und Log-Level für die Abonnements anzupassen [TKG21a, S. 22][TKG21b, min. 19:29].

Generell wird meistens beim Vorgang, die im ETW-Provider erzeugten Event-Objekte an das ETW-System zu transferieren, die System-Dispatch-Schnittstelle mit den Funktionen `ntoskrnl.exe!NtTraceEvent` und `ntoskrnl.exe!ZwTraceEvent` verwendet. Aus dem User-Mode führt der Systemcall durch `NTDLL.dll!NtTraceEvent` ausgelöst zum Aufruf von `ntoskrnl.exe!NtTraceEvent`. Im Kernel-Mode kann der ETW-Provider-Code `ntoskrnl.exe!ZwTraceEvent` verwenden. Würden diese Funktionen gehookt oder gepatcht werden, könnte der Kontrollfluss eventuell so verändert werden, dass die Event-Objekte global nicht mehr an das ETW-System weitergeleitet würden. Dies würde zum systemweiten Ausfall des Event-Logging mittels ETW führen. Da entsprechende Code- und Datenbereiche des System-Dispatch-Service durch PatchGuard (KPP) geschützt sind, müssten hier komplexe Hooking-Methoden wie InfinityHook zum Einsatz kommen. InfinityHook ist ein Rootkit-Werkzeug, das aus dem Kernel-Mode heraus angewendet werden kann und es ermöglicht, System-Calls, Kontextwechsel, Page-Fault-Exceptions, DPCs und andere Systemereignisse zu instrumentieren, ohne das dies von PatchGuard (KPP) oder HyperGuard (VBS) erkannt wird [EMY19][TKG21a, S. 22][TKG21b, min. 19:29].

Da die Datenstrukturen des ETW-Systems nicht durch PatchGuard (KPP) geschützt sind, können auch diese durch einen im Kernel-Mode agierenden Schadcode gesucht und verändert werden, um die internen Vorgänge zu manipulieren. Das ETW-System befindet sich in einem Silo-Kontext einer Art Container des Windows-eigenen Silo-Container-Systems. In der Datenstruktur `_ETW_SILODRIVERSTATE` können die Felder `MaxLoggers`, `EtwpSecurityLoggers` und `SystemLoggerSettings` geändert werden,

um das ETW-System zu stören. `_ETW_SILODRIVERSTATE` kann über eine Adresszeigerkette ausgehend von der exportierten Datenstruktur `_ESERVERSILO_GLOBALS` erreicht werden (`ntoskrnl.exe!PspHostSiloGlobals : _ESERVERSILO_GLOBALS -> EtwSiloState : _ETW_SILODRIVERSTATE`). Die Eigenschaften beziehen sich auf das gesamte ETW-System des Silos. Solange keine Container-Funktionalität aktiviert ist, existiert das ETW-System einmal systemweit im Kernel. Es kann jedoch auch mehrere Instanzen für verschiedene Silos geben, wenn die Container-Funktionalität des Windows-Kernels genutzt wird. Durch das Ändern des Wertes in `MaxLoggers` kann die Anzahl der maximal erlaubten ETW-Sessions herabgesetzt werden. Dies könnte dazu benutzt werden, eine Sicherheitssoftware daran zu hindern, eine neue ETW-Session zu starten. Das Feld `SystemLogger` spezifiziert die Eigenschaften von System-Sessions [TKG21a, S. 23][TKG21b, min. 20:21].

Das Feld `EtwpLoggerContext` der `_ETW_SILODRIVERSTATE`-Struktur enthält einen Adresszeiger auf die Liste der aktiven ETW-Sessions. Diese ist ein Array fester Länge mit den Adresszeigern auf die zentrale Verwaltungsstruktur `WMI_LOGGER_CONTEXT` der verschiedenen aktiven ETW-Sessions. Jeder Eintrag steht für eine Session, die jeweils eine eigene `WMI_LOGGER_CONTEXT`-Struktur besitzt. Nicht belegte Plätze im Array besitzen einen Null-Pointer. Durch das Überschreiben bestimmter Einträge des Arrays mit Nullen, würden entsprechende ETW-Sessions aus dem ETW-System entfernt und könnten keine Ereignisse mehr aufzeichnen [TKG21a, S. 23][TKG21b, min. 20:21].

In der Datenstruktur `WMI_LOGGER_CONTEXT` einer ETW-Session befinden sich die Felder, deren Werte die ETW-Session spezifizieren. Durch verändern der Werte, können die Eigenschaften der ETW-Session geändert werden. Durch ändern der Felder `LoggerId`, `LoggerMode` und `LoggerName` kann die Session-Instanz geändert werden, sodass sie nicht mehr von den Controller- und Consumer-Applikationen erreicht werden kann. Im Feld `Flags` gibt es das Bit `SecurityTrace`. Dieses bestimmt darüber, ob Controller- und Consumer-Prozesse im PPL-Modus (Protected Process Light) ausgeführt werden müssen und einer digitalen Signatur bedürfen, um auf die Session zugreifen zu dürfen. Durch eine Änderung kann diese Restriktion entfernt werden. Das Feld `SecurityDescriptor` enthält die Security-Deskriptor-Datenstruktur, die serialisiert der Bytefolge in der Registry entspricht, in der die Deskriptoren persistent gespeichert werden. Änderungen an den beiden zuletzt genannten Attributen kann die Zugriffsrichtlinien für die ETW-Session ändern, sodass bösartigen Prozessen der Zugriff auf die Session gewährt werden kann [TKG21a, S. 23][TKG21b, min. 20:21][TKG21a, S. 33ff][TKG21b, min. 29:19][AIRS21, S. 522ff]. Die `WMI_LOGGER_CONTEXT`-Struktur wurde bereits „reverse engineered“, sodass die Offsets sehr vieler Felder der Struktur bekannt sind [EMY19].

3.9.2.8 Angriffe über CVE-Schwachstellen

Im Laufe der Zeit wurden immer wieder Schwachstellen in den verschiedensten Teilen des ETW-Systems bekannt. Manche bezogen sich z.B. auf die Kommunikationsschnittstelle zwischen der ETW-API und dem ETW-System im Kernel. Im speziellen gab es Sicherheitslücken in der Funktion `NtTraceControl` (CVE-2020-1034, EtwReceiveNotification). Aber auch in anderen Schnittstellen und Konfigurations-Parametern des ETW-Systems wurden Schwachstellen gefunden, über die sich Event-Tracing-for-Windows (ETW) angreifen lässt [Sha20].

Sollte der Angreifer Wissen über solche Sicherheitslücken im ETW-System besitzen, die zum Zeitpunkt des Angriffs nicht geschlossen sind, könnte Schadsoftware diese ausnutzen, um das ETW-System zu manipulieren, die Event-Protokollierung zu stören oder anderweitig über diese Wege Schaden anzurichten.

3.9.2.9 Störung von ETW und EDR-Sensoren durch Überlastung und Fake-Ereignisse

Die bisher beschriebenen Angriffe auf die Ereignisüberwachung mittels ETW zielten überwiegend darauf ab, die Event-Erzeugung zu stören oder ganz zu deaktivieren. Ein alternativer Ansatz, Sicherheitssoftware oder EDR-Lösungen zu stören, könnte darin bestehen, die ETW-Consumer-Software mit großen Mengen an Ereignis-Signalisierungen zu überlasten oder falsche Ereignisse zu signalisieren. Letzteres könnte dazu führen, dass Sicherheitssysteme eventuell abgeschaltet werden, da sie den Betrieb stören.

4 Sysmon

Die Software Sysmon ist ein von Mark Russinovich (Windows Internals) und Thomas Garnier entwickeltes Monitoring-Werkzeug der Windows-Sysinternals-Suite. Die Software beinhaltet eine im Hintergrund laufende Dienstanwendung und einen von Microsoft signierten Kernel-Mode-Treiber. Das Kernel-Modul erkennt Systemereignisse und meldet diese dem User-Mode-Dienst, der diese als Event-Objekte in den Windows-Eventlog schreibt. Bei der Auswahl der von Sysmon erkannten Systemereignisse liegt der Fokus auf sicherheitsrelevanten Verhaltensereignissen, die sich auf das Zugriffsverhalten von Prozessen beziehen. Aus diesem Grund wird Sysmon häufiger als Sensor in Telemetrie-Lösungen eingesetzt. Die im Event-Log gesammelten Sysmon-Ereignisse vieler Endpunkte werden dabei zu einer zentralen Stelle übertragen und dort ausgewertet. Auf den Ereignissen wird versucht, bösartiges Verhalten auf einzelnen Maschinen zu erkennen und darauf zu reagieren. Sysmon liefert nur Ereignisse, die das Verhalten von einzelnen Prozessen beschreiben. Dabei handelt es sich nicht um feingranulare Einzelereignisse, sondern um konkrete Aktionen, die aus bereits aggregierten Einzelereignissen abgeleitet werden. Insgesamt handelt es sich dabei um 26 Ereignistypen, die sicherheitsrelevante Vorgänge im Prozessverhalten beschreiben [Mon22][Har21a][Har21b]. Jedes Event-Objekt besitzt eine Reihe von Attributen. Ein Teil dieser Attribute geht über die technisch notwendigen Parameter, die eine Verhaltens-Aktion beschreiben, hinaus und reichern das Event-Objekt mit zusätzlichen Informationen an, die für die weitere Verarbeitung sehr nützlich sind. Diese Informationen beschreiben meist den verursachenden Prozess und die Systemressource, auf die zugegriffen wird. Beispiele sind Hash-Werte von Dateien und geladenen Modulen, Datei- und Objekt-Pfade, sowie GUIDs für Prozesse und Threads, um diese eindeutig zu unterscheiden [LB19, S. 3][Mic1a][Mon22][PHAK21].

Die Sysmon-Sensorik basiert auf Kernel-Callbacks, Filtertreibern, WMI-Filtern und Event-Tracing-for-Windows (ETW). Wobei der Großteil der Ereignisse auf Basis der ersten beiden genannten Technologien erfolgt. Alle Verfahren sind im ersten Kapitel dieser Arbeit im Abschnitt „Sensorik“ erläutert [LB19, S. 13][Mon22].

4.1 Funktionalität

Die Sysmon-Software beinhaltet ein Anwendungsprogramm (`sysmon.exe`). Diese Anwendung bringt die Funktionalität mit, Sysmon als Dienst auf dem System zu installieren und zu deinstallieren. Aufgerufen mit dem Parameter „`-i`“, installiert die Anwendung sich selbst als Windows-Dienst, der automatisch beim Systemstart gestartet wird und dauerhaft im Hintergrund ausgeführt wird. Zudem wird der Sysmon-Kernel-Treiber (`SysmonDrv.sys`) installiert, sodass dieser beim Booten in den Windows-Kernel geladen

wird. Als weiterer Parameter kann bei der Installation die Konfigurationsdatei angegeben werden, über die unter anderem die anzuwendenden Filterregeln bestimmt werden. Sysmon erkennt durch den Kernel-Mode-Treiber spezielle Ereignisse des Systemverhaltens. Die Ereignisse werden als Sysmon-Event-Objekte durch die Dienstanwendung in eine spezielle Sysmon-Kategorie in den Windows-Event-Log geschrieben. Welche erkannten Ereignisse Sysmon als Event-Objekt in den Event-Log schreibt, hängt von den konfigurierten Filterregeln ab. Die protokollierten Sysmon-Ereignisse können im Event-Log unter `/Applications and Services Logs/Microsoft/Windows/Sysmon/Operational` gefunden werden [Mic1a].

4.1.1 Regelbasierte Ereigniserkennung

Sysmon protokolliert nur dann Ereignisse im Event-Log, wenn eine konfigurierte Regel die Protokollierung des Ereignisses einschließt. Somit kann der Output der Sysmon-Software durch den konfigurierten Regelsatz weitreichend gesteuert werden. Die Regel-Konfiguration wird in der Sysmon-Konfigurationsdatei im XML-Format spezifiziert. Für jeden Ereignistyp können zwei Regelsätze als XML-Knoten erstellt werden. Einen Include-Regelsatz, der beschreibt, bei welchen Bedingungen das Ereignis in den Event-Log geschrieben werden soll. Und einen Exclude-Regelsatz, der beschreibt, wann ein Ereignis nicht geloggt werden soll. In den XML-Knoten der Regelsätzen können die einzelne Regeln als weitere XML-Knoten spezifiziert werden. Jede Regel wird durch einen XML-Knoten beschrieben und spezifiziert einen Wertefilter bezogen auf ein konkreten Attribut des Ereignistyps. Für ein Attribut können mehrere Regeln spezifiziert werden. Bei Konflikten zwischen dem Include-Regelsatz und dem Exclude-Regelsatz hat der Exclude-Regelsatz Vorrang [LB19, S. 5][Mic1a][PHAK21].

Über die Sysmon-Konfiguration wird meist versucht die Ereignisse von bekanntem Systemverhalten herauszufiltern, sodass die Menge an zu protokollierenden Ereignissen sinkt. Beispielsweise werden Systemprozesse und Hintergrunddienste oft aus dem Logging ausgeschlossen. Im Gegenzug werden bestimmte Artefakt-Muster, die Malware bekanntermaßen verwendet, als Regeln definiert, um Ereignisse, in denen diese vorkommen, einzuschließen. Beispiele sind hier unter anderem Muster für Dateinamen, Dateispeicherorte (Pfadnamen), Registry-Schlüssel oder Prozessnamen. Zum Teil werden vermeintlich gute Sysmon-Konfigurationen unter Sicherheitsforschern und Systemadministratoren ausgetauscht und öffentlich bereitgestellt. Die Sysmon-Konfiguration von SwiftOnSecurity ist in dem Kontext zu nennen, die öffentlich über GitHub bereitgestellt wird und in vielen Organisationen als Sysmon-Konfiguration eingesetzt wird [LB19, S. 36ff][PHAK21].

4.1.2 Prüfsummenbildung

Sysmon kann über bestimmte mit den Ereignissen assoziierte Daten Prüfsummen bilden. Der Mechanismus kann aktiviert und konfiguriert werden, jedoch auch abgeschaltet sein. Anwendung findet die Prüfsummenbildung für den Inhalt des Executables aus dem bei der Prozesserstellung, der neue Prozess instanziiert wird, sowie für den Inhalt von Modulen und Kernel-Treiber, die als Images in Adressräume geladen werden. Als Hash-Algorithmus können dabei ein oder mehrere der vier Verfahren `SHA1`, `MD5`, `SHA256` und `IMPHASH` genutzt werden. Es können wie erwähnt auch mehrere der vier Hash-Verfahren parallel genutzt werden. Die Hash-Werte der ausgewählten verfahren werden durch Komma getrennt hintereinander in die Zeichenkette des Attributes `Hashes` im Event-Objekt eingetragen [Mic1a][PHAK21].

4.1.3 Reverse DNS-Lookup

Sysmon kann so konfiguriert werden, dass es für Netzwerk-Ereignisse die Daten der Event-Objekte, die standardmäßig IP-Adressen enthalten, mit DNS-Namen anreichert, die mit den IP-Adressen assoziiert sind. Dafür stellt die Sysmon-Dienstanwendung Reverse-DNS-Anfragen, um mögliche DNS-Namen für die IP-Adressen zu ermitteln. Die DNS-Namen werden im Sysmon-Dienst gepuffert, sodass immer nur für neue, noch unbekannte IP-Adressen Namen ermittelt werden [Mic1a][PHAK21].

4.1.4 Archivierung

Sysmon bringt die Funktionalität mit, bestimmte Daten in Dateien in einem Archiv-Verzeichnis zu sichern, direkt bevor diese überschrieben werden. So können über die Konfigurationsdatei Dateien und Verzeichnisse festgelegt werden, bei denen Sysmon beim Feststellen eines Löschvorgangs den Inhalt in entsprechende Dateien im, über die Konfiguration spezifizierten, Archiv-Verzeichnis gesichert werden, bevor der Löschvorgang durchgeführt wird. Ähnliches ist auch für Inhalte der Zwischenablage möglich, sodass jeder Inhalt in der Zwischenablage geloggt werden kann. An dieser Stelle wird deutlich, das Sysmon durch die Kernel-Callbacks und Filter-Callbacks der Filter-Treiber innerhalb der zu messenden Vorgänge synchron eigenen Code ausführen kann. Damit kann Sysmon Vorgänge selbst unterbrechen und eigene Aktionen durchführen. Dies sind Dinge, die ein rein auf ETW basierendes System nicht liefern kann, da dort die Ereignisse erst nachdem diese passiert sind, asynchron beim Consumer eingehen [Mic1a][PHAK21].

Durch die Konfigurationsdatei können folgende Archivierungsvorschriften für Dateilöschungen eingestellt werden:

Konfigurationsparameter	Typ	Beschreibung
`CopyOnDeletePE`	`Boolean`	gibt an, ob PE-Dateien vor ihrer Löschung gesichert werden sollen
`CopyOnDeleteSIDs`	`String`	Liste mit Benutzerkonton, deren Löschvorgänge zu einer vorherigen Sicherung der Dateien führen
`CopyOnDeleteExtensions`	`String`	Liste mit Dateinamen-Erweiterungen, deren Dateien vor einem Löschvorgang im Archiv gesichert werden
`CopyOnDeleteProcesses`	`String`	Liste mit Prozesses, deren Löschvorgänge zu einer vorherigen Sicherung der Dateien führen

Tabelle 4.1: Sysmon: Archivierungsvorschriften für Dateilöschvorgänge [Mic1a]

4.1.5 Wiederherstellung

Als Ergänzung zur Archivierung kann Sysmon so konfiguriert werden, dass für bestimmte konfigurierte Objekte wie z.B. Registry-Einträge Änderungsvorgänge eine automatische Wiederherstellung auslösen. So würde Sysmon direkt nach dem erkannten Änderungsvorgang den geänderten oder gelöschten Registry-Schlüssel oder das geänderte oder gelöschte Regsitry-Werte-Feld wiederhergestellen. Dadurch wird es möglich, bestimmte Bereiche der System-Konfiguration vor Veränderungen durch Schadsoftware zu schützen [Mic1a][PHAK21].

4.1.6 Früher Start

Die Sysmon-Dienstanwendung kann als Windows-Dienst installiert werden und so konfiguriert werden, dass diese automatisch bei der Systeminitialisierung gestartet wird. Darüber hinaus kann der Dienst besonders früh vor den meisten anderen Diensten gestartet werden und der Kernel-Driver kann noch vor der Dienstanwendung im Bootprozess geladen werden. Letzteres ermöglicht Sysmon Vorgänge im Boot-Prozess und während der Systeminitialisierung zu erkennen, die es möglich machen in manchen Fällen Kerlen-Mode-Malware wie Rootkits zu erkennen [Mic1a][PHAK21].

4.1.7 Obfuskierung der Sysmon-Software

Sysmon bietet die Möglichkeit die Namen des Dienstprozesses und des Kernel-Mode-Treibers anzupassen. Dies erschwert es Malware das Vorhandensein einer Sysmon-Sensorik zu detektieren [Mic1a][PHAK21].

4.2 Eigenschaften

In diesem Abschnitt wird auf die drei Basiseigenschaften Ausführungskontext, Auftrittszeit und Abstraktionsebene, die im ersten Kapitel allgemein eingeführt und erläutert wurden, Bezug genommen und geprüft, wie diese durch die Monitoring-Software Sysmon umgesetzt sind.

4.2.1 Ausführungskontext

Fast alle Sysmon-Ereignisse besitzen Attribute, die den Prozess, der das Ereignis verursacht hat, identifizieren und beschreiben. Eindeutig Identifiziert werden kann der Prozess über das Attribute `ProcessGUID`, welches eine GUID enthält. Darüber hinaus besitzen die meisten Ereignistypen noch weitere beschreibende Attribute, wie die Prozess-ID unter Windows, den Dateipfad des Executables und das Benutzerkonto, unter dem der Prozess ausgeführt wird, liefern. Es gibt Ausnahmen unter den Ereignistypen, die keine Information über den Prozess-Kontext, indem das Ereignis aufgetreten ist, liefern. Die Ausnahmen sind die WMI-Ereignisse, Kernel-Driver-Load-Ereignisse und die Ereignisse, die den Sysmon-Dienst betreffen [Mon22][PHAK21][Har21a].

Ein Referenz auf den Thread, der das Ereignis ausgelöst hat, ist bei Sysmon bis auf eine Ausnahme nicht Teil der Event-Attribute. Der Ereignistyp, der signalisiert, dass ein Prozess auf ein fremdes Prozess-Objekt zugreift, beinhaltet als einziger Eventtyp in seinen Attributen die ID des auslösenden Threads.

Der CPU-Kern, auf dem ein Ereignis ausgelöst wurde, ist bei Sysmon bei keinem Ereignistyp Teil der Event-Informationen.

4.2.2 Auftrittszeit

Die Ereignisse, die Sysmon liefert, zeigen in jedem Fall an, dass ein bestimmter Verhaltensvorgang durchgeführt wurde. Eine Signalisierung, bevor bestimmte Aktionen durchgeführt werden, bietet Sysmon nicht. Dabei werden auch nur Aktionen registriert, die erfolgreich abgeschlossen wurden. Gescheiterte Versuche eines Prozesses auf bestimmte Systemressourcen zuzugreifen, werden von Sysmon generell nicht als Event-Objekte geliefert. Der Zeitpunkt, an dem das Sysmon-Event-Objekt erzeugt wird, liegt somit immer hinter dem eigentlichen Zeitpunkt des Ereignisses, dessen erfolgreiche Durchführung das Sysmon-Event signalisiert [Mon22][PHAK21].

Sysmon protokolliert in jedem erzeugten Event-Objekt den Zeitpunkt, zudem das Ereignis aufgetreten ist bzw. zudem der Vorgang erfolgreich abgeschlossen wurde. Der Zeitpunkt wird im Attribut `UtcTime` eines jeden Event-Objektes als Unicode-Zeichenkette gespeichert. Die Zeichenkette besitzt das Format `YYYY-MM-DD hh:mm:ss.sss` (Beispiel: `2022-01-03 14:36:06.768`). Der Zeitpunkt wird in Realzeit, bezogen auf die Koordinierte-Weltzeit (UTC), mit der zeitlichen Auflösung von einer Millisekunde angegeben [Mic1a][Mon22][PHAK21].

4.2.3 Abstraktionsebene

Wie erwähnt sind die Ereignisse, die Sysmon liefert, allesamt spezielle Aspekte, die das Verhalten von Prozessen beschreiben. In der Regel handelt es sich dabei um Zugriffs-Ereignisse, die ein Prozess auf bestimmte Systemressourcen durchführt. Dabei steckt die Aussagekraft der von Sysmon gelieferten Ereignisse darin, zu beschreiben, dass ein bestimmter Zugriff eines Prozesses auf eine Ressource stattgefunden hat. Wie der Zugriffsvorgang im Detail abgelaufen ist, steht nicht im Fokus. So liefert Sysmon keine elementaren Einzelanfragen, die der Prozess an den Betriebssystem-Kernel stellt wie z.B. das Öffnen einer Datei und das Schreiben eines Pufferinhaltes in eine Datei. Stattdessen werden zusammengefasste Zugriffsvorgänge als Event-Objekte geliefert wie beispielsweise das Ändern eines Dateiinhaltes durch einen bestimmten Prozess. Die Abstraktionsebene ist somit die „aggregierte Prozessverhaltensebene“ bezugnehmend auf den Abschnitt „Abstraktionsebenen“ des ersten Kapitels.

4.3 Architektur

Die Sysmon-Software besteht aus zwei Komponenten einer User-Mode-Applikation (`Sysmon.exe`), die als Windows-Service ausgeführt wird, und einem Kernel-Mode-Treiber (`SysmonDrv.sys`). Die Dienstanwendung veranlasst auf Anweisung oder automatisch beim Starten das Laden des Kernel-Moduls. Beide Instanzen kommunizieren über die Geräteschnittstelle. Die Kommunikation läuft dabei schlussendlich über die Funktion `NtDeviceIoControl` der `NTDLL.dll`, die die Systemcalls tätigt, die in den IO-Manager des Kernels verweisen. Die Anfragen werden aufgrund des gewählten Device-Objektes zum geladenen Treiber `SysmonDrv.sys` geroutet [LB19, S. 6][LB19, S. 13][Mon22].

Darüber hinaus erstellt die Dienstanwendung eine System-Session im ETW-System und abonniert die System-Provider-Klasse „NT-Kernel-Trace“. Die Provider-Enable-Flags werden so gewählt, dass die entsprechenden Ereignistypen empfangen werden. Dies sind hauptsächlich Ereignisse bezüglich der Netzwerkaktivität (TCP/IP-Stack). Für die Erkennung der meisten anderen Vorgänge wird nicht ETW genutzt. Die Session wird

als Real-Time-Trace-Session konfiguriert, sodass aus ihr direkt Event-Objekte gelesen werden können und eine Benachrichtigung stattfindet, wenn neue Event-Objekte in den Puffern eingehen. Die Dienstanwendung konsumiert selbst die anfallenden ETW-Events der von ihr gestarteten System-Session und erstellt daraus Sysmon-Event-Objekte für den Windows-Event-Log. Sie übernimmt somit im ETW-Kontext sowohl die Rolle des ETW-Controllers als auch die Rolle des ETW-Consumers [LB19, S. 32ff][LB19, S. 13][Mon22].

Um sicherheitsrelevante Aktionen des WMI-Systems zu erkennen, nutzt Sysmon nicht ETW sondern spezielle WMI-Filter. Die dadurch ermittelten Ereignisse resultieren in den drei sicherheitsrelevanten WMI-Ereignistypen (19, 20, 21). Für die WMI-Filterung installiert die Dienstanwendung in der WMI-Konfiguration Filter-Routinen für drei WMI-Klassen. Der WMI-Dienst-Prozess (`WmiPrvSe.exe`), der auch WMI-Provider-Host genannt wird, führt die Filter-Routinen aus. Die Filter-Routinen werden alle fünf Sekunden aufgerufen und versuchen aus den Daten bestimmte Zugriffe zu erkennen. Eine Erkennung wird der Sysmon-Dienstanwendung vom WMI-Provider-Host über die WMI-Service-Schnittstelle (Interprozesskommunikation) zurückgemeldet [LB19, S. 28ff][LB19, S. 13][Mon22].

Das geladene Kernel-Modul (`SysmonDrv.sys`) registriert bei seiner Initialisierung viele verschiedene Callback-Routinen für eine ganze Reihe von Kernel-Benachrichtigungsfunktionen (Kernel-Callbacks). Über diese Mechanismen werden die registrierten Routinen in `SysmonDrv.sys` aufgerufen, wenn bestimmte Ereignisse im Windows-Kernel auftreten (`ntoskrnl.exe`). Die Callback-Routine wird dabei synchron im Kontext des Prozesses, der die zu erkennende Aktion durchführt, aufgerufen [LB19, S. 22][LB19, S. 21][LB19, S. 23][LB19, S. 13][Mon22]. Die Aktion wird im Prozesskontext des verursachenden Prozesses erkannt und nach einem Thread-Wechsel im Prozesskontext der, auf Ereignisse vom Kernel-Treiber wartenden, Dienstanwendung über die Geräteschnittstelle mitgeteilt[LB19, S. 19f].

Für nicht alle Vorgänge und Zugriffe bietet der Windows-Kernel eine Benachrichtigungsfunktionalität (Kernel-Callbacks) über zu registrierende Kernel-Callback-Routinen an. Für solche Gebiete der Verhaltenserkennung, für die keine Callback-Funktionen registriert werden können, verwendet der Sysmon-Treiber die Filter-Funktionalitäten des IO-Managers und des Objekt-Managers. So installiert das Sysmon-Kernel-Modul eigene Filter-Geräte beim Filter-Manager für die Überwachung der Nutzung von Named-Pipes, Objekt-Zugriffen auf Thread-Objekte, Prozess-Objekte und Desktop-Objekte sowie Zugriffe auf Dateien und Verzeichnisse auf Datenträgern (File-System-Filter-Driver) [LB19, S. 24ff][LB19, S. 27][LB19, S. 25ff]. Dabei werden bei aktuellen Versionen von Sysmon und Windows die neueren Mini-Filter verwendet. Dies hat den Vorteil, dass nicht für jeden Filter ein separates Filtergeräte-Objekt erstellt werden muss. Stattdessen kann der

Sysmon-Treiber eigene Callback-Routinen als Filter-Routinen beim Filtermanager registrieren, die von diesem aufgerufen werden. In jedem Fall wird der Sysmon-Treiber selbst zum Filtertreiber. Es können pro Filter-Stelle zwei Routinen registriert werden. Die Pre-Operation-Routine wird aufgerufen, wenn die Anfrage oder im Falle von IO-Zugriffen das IRP-Paket den Geräte-Stapel hinab gereicht wird, und ermöglicht es, die Anfrage zu filtern bzw. diese zu behandeln oder zu bearbeiten. Die Post-Operation-Routine wird aufgerufen, wenn der Zugriff erfolgt ist und die Antwort an die anfragende-Komponente zurückgeliefert wird, und ermöglicht ebenfalls, diese Antwort zu behandeln oder zu bearbeiten [LB19, S. 8ff][Yos20, S. 243ff][Yos20, S. 159ff][Yos20, S. 166ff]. Sysmon nutzt für die Erkennung eines erfolgreichen Vorgangs nur die Post-Operation-Routine, über die erkannt werden kann, das die entsprechenden Vorgänge abgeschlossen wurden. Die Sysmon-Ereignisse beschreiben daher immer das bestimmte Vorgänge geschehen sind. Ereignisse, die implizieren, dass Vorgänge geschehen werden, liefert Sysmon prinzipiell nicht [LB19, S. 24ff][LB19, S. 27][LB19, S. 25ff]. Die erkannten Ereignisse teilt das Sysmon-Kernel-Modul der Sysmon-Dienstanwendung über die eigene Geräteschnittstelle mit[LB19, S. 19f].

Alle in der Dienstanwendung eingehenden Ereignisinformationen aus den verschiedenen Quellen (ETW, WMI-Filter, Kernel-Callbacks und Filtertreiber) werden in der Dienstanwendung aufbereitet und mit Informationen angereichert. Einige der Informationen stammen aus dem Mitverfolgen bestimmter Vorgänge und dem Führen von Zuordnungstabellen in der Sysmon-Dienstanwendung. So ergänzt Sysmon z.B. die Attribute des Ausführungskontext zusätzlich zur Prozess-ID mit einer entsprechenden GUID. Zusätzlich wird der Benutzerkontokontext als SID und der Pfadname des Executables als weitere Eigenschaften des Prozesses angegeben. Darüber hinaus ermittelt die Dienstanwendung unter anderem Pfadnamen zu Schlüssel- und Dateiobjekten und verfolgt diese, um die Informationen in den Attributen der Event-Objekte zu hinterlegen. Zudem kann ein Hash-Algorithmus (`SHA1`, `MD5`, `SHA256`, `IMPHASH`) spezifiziert werden, mit dem Sysmon für den Inhalt von Dateien eine Prüfsumme bildet und diese in den Attributen bestimmter Ereignistypen ergänzt. Die Dienstanwendung führt für IP-Adressen von Neztwerk-Ereignissen Reverse-DNS-Anfragen durch und puffert diese, um den Event-Objekten, die Internet-Aktivitäten melden, neben den IP-Adressen auch entsprechende Domain-Namen als Attribute bereitzustellen. Zusammenfassend entstehen in der Sysmon-Dienstanwendung angereicherte Sysmon-Event-Objekte, die vom Service in den Windows-Event-Log geschrieben werden [LB19, S. 6][LB19, S. 19f].

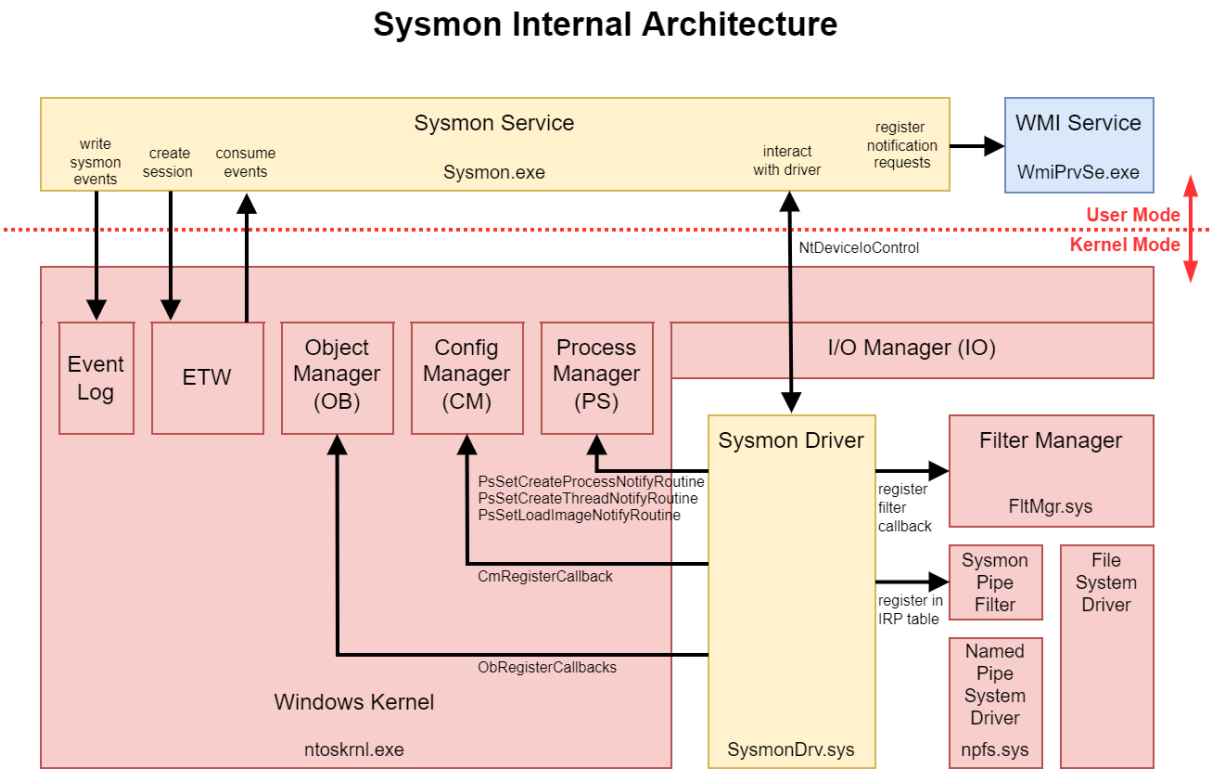

Abbildung 4.1: Sysmon - Architektur

4.3.1 Erkennung von Prozess- und Thread-Erzeugungen

Das Erzeugen von Prozessen und Threads wird im Sysmon-Kernel-Modul (`SysmonDrv.sys`) erkannt. Die Erkennung basiert auf zwei Callback-Routinen des Sysmon-Treibers, die über die Funktionen `PsSetCreateProzessNotifyRoutine` und `PsSetCreateThreadNotifyRoutine` beim Windows-Kernel (`ntoskrnl.exe`) registriert werden. Der Kernel ruft diese Routinen nach dem Erstellen eines neuen Prozesses oder Threads auf, bevor die Kontrolle aus dem Kernel an den beauftragenden User-Mode-Code zurückgegeben wird [LB19, S. 22][Mon22].

4.3.2 Erkennung von Abbild-Ladevorgängen

Das Erkennen von Ladevorgängen von Modulen (DLL-Bibliotheken, Kernel-Treiber, andere ladbare Module) wird auch durch den Sysmon-Kernel-Treiber ermöglicht. Der Treiber registriert über die Funktion `PsSetLoadImageNotifyRoutine` eine eigene Erkennungsfunktion als Callback-Funktion beim Windows-Kernel (`ntoskrnl.exe`). Die Funktion wird vom Loader im Windows-Kernel aufgerufen, nachdem ein Modul geladen wurde. Dabei kann es sich sowohl um eine Bibliothek oder andere Ressource, die in den User-Space-Adressraum des Prozesses geladen wurde, handeln, wie z.B. eine DLL-Bibliothek,

als auch um ein Kernel-Modul, dass in den allgemeinen Kernel-Space-Adressraum geladen wurde. Diese Callback-Funktionalität ermöglicht Sysmon somit die Sysmon-Events „ImageLoad“ (7) und „DriverLoad“ (6) zu erkennen [LB19, S. 21][Mon22].

4.3.3 Erkennung von Registry-Zugriffen

Registry-Zugriffe werden ebenfalls durch eine spezielle Callback-Routine des Sysmon-Drivers, die dieser beim Windows-Kernel (`ntoskrnl.exe`) registriert, erkannt. In diesem Fall wird die Routine nicht beim Prozess-Manager (PS) sondern beim Config-Manager (CM) des Kernels registriert. Dafür ist eine Funktion vorgesehen, mittels der eine Routine für verschiedene Config-Manager-Aktionen registriert werden kann. Die Routine kann intern anhand der Parameter auswerten, um welche Aktion es sich handelt. Der Sysmon-Treiber benutzt diese `CmRegisterCallback`-Funktion, um seine Erkennungs-Funktion als Callback-Routinen zu installieren, um über verschiedene Zugriffsereignisse auf Registry-Elemente (Schlüssel, Werte-Felder) benachrichtigt zu werden und diese somit zu erkennen. Neben `CmRegisterCallback` besteht eine weitere Funktion `CmRegisterCallbackEx`, die laut Microsoft-Dokumentation für Treiberentwickler ab Windows 7 statt der älteren Variante verwendet werden soll, die aber nach wie vor noch zur Verfügung steht [LB19, S. 23][Mon22].

4.3.4 Erkennung von Zugriffen auf Prozess-Objekte, Thread-Objekte und Desktop-Objekte

Prozesse, Threads und Desktops werden neben vielen anderen Systemressourcen im Windows-Kernel als System-Objekte abgebildet. Die System-Objekte liegen in einem eigenen Namensraum (Object-Pool), der vom Objekt-Manager (OB) des Kernels verwaltet wird. Der Sysmon-Treiber registriert für die Erkennung von Zugriffen auf Prozess-, Thread- und Desktop-Objekte Callback-Routinen beim Objekt-Manager (OB). Dafür verwendet das Sysmon-Kernel-Modul die Funktion `ObRegisterCallbacks`, die vom Windows-Kernel (`ntoskrnl.exe`) exportiert wird. Die Funktion erwartet als Parameter einen Zeiger auf eine Struktur, die eine Reihe von Funktionspointern auf verschiedene Callback-Routinen enthalten kann. Sysmon registriert Callback-Routinen, die aufgerufen werden, wenn ein Prozess-Objekt, Thread-Objekt oder Desktop-Objekt von einem Prozess geöffnet wurde und wenn ein Objekt der entsprechenden Typen geändert wurde. Dabei registriert Sysmon nur Post-Operation-Callbacks, deren Aufruf durch den Objekt-Manager nur das erfolgreiche Ende der jeweiligen Einzelaktion signalisiert. Dadurch erkennt der Sysmon-Treiber die Zugriffe auf diese Ressourcen und meldet dies der Dienstanwendung [LB19, S. 27][Mon22].

4.3.5 Erkennung von Zugriffen auf das Dateisystem

Um Zugriffe auf Dateien und Verzeichnisse auf Datenträgern zu erkennen, nutzt Sysmon die Filter-Treiber-Funktionalität des IO-Managers. Logische Datenträger wie z.B. das Laufwerk `C:\` sind unter Windows durch Datenträger-Geräte-Objekte abgebildet, die entsprechende logische oder physikalische Partitionen auf realen physischen Datenträgern oder Datenträgerverbünden als „logisches Laufwerk“ beschreiben. Bearbeitet werden Anfragen auf diese logischen Datenträger-Objekte von den mit diesen Objekten assoziierten Dateisystem-Treibern wie z.B. dem NTFS-Treiber. In Kernel-Versionen vor Windows Vista (6.0) legte der Sysmon-Treiber für jeden logischen Datenträger ein neues Filter-Geräte-Objekt beim IO-Manager an und positionierte diese im Gerätestapel oberhalb der Geräte-Objekte der logischen Datenträger (`\Device\HardDdiskVolume*`), sodass Anfragen auf die logischen Datenträger zuerst Code im Sysmon-Treiber hinter den neu erstellten Geräte-Objekten durchlaufen mussten. Über die IRP-Tabellen (IRP: IO Request Paket) der Geräteobjekte registrierte sich der Sysmon-Kernel-Mode-Treiber selbst als Filtertreiber, indem er für verschiedene Aktionen des Geräte-Objektes eigene Callback-Routinen spezifizierte. Durch diese Filterroutinen konnte der Sysmon-Treiber Zugriffe auf Dateien und Verzeichnisse erkennen, deren Argumente dem IRP-Paket entnehmen und die Vorgänge der Dienstanwendung melden [LB19, S. 8ff][LB19, S. 10ff][LB19, S. 25ff].

Seit Windows Vista ($>=$ 6.0) wird statt der Registrierung spezieller Filter-Geräte-Objekte beim IO-Manager die Mini-Filter-Funktionalität des Filter-Managers genutzt. Dabei registriert der Sysmon-Treiber spezielle Filterroutinen direkt beim Filter-Manager für die Filterung von Anfragen auf Datenträger-Geräte-Objekte. Der Filtermanager ruft die registrierten Callback-Routinen auf, wenn auf die logischen Datenträger also auf NTFS-Dateisysteme zugegriffen wird. Die Anfragen werden mit dem entsprechenden IRP-Paket von den Callback-Routinen des Sysmon-Treiber behandelt, bevor diese an den Dateisystemtreiber weitergereicht werden. Auf diese Weise können die Zugriffe auf Dateien und Verzeichnisse in NTFS-Partitionen vom Sysmon-Treiber erkannt werden und an den Dienstprozess gemeldet werden [LB19, S. 25ff].

4.3.6 Erkennung von Interprozesskommunikation und Service-Anfragen

Eine wichtige Kernel-Funktionalität für Interprozesskommunikation sind Named-Pipes. Die Kommunikation zwischen vielen Anwendungsprozessen und den Windows-Diensten wird z.B. oft über Namped-Pipes realisiert. Named-Pipes unter Windows sind System-Objekte im „Named-Pipe-Dateisystem“ (Namensraum für Pipe-Objekte), das vom

Windows-Treiber `NPFS.sys` (Named-Pipe-File-System) implementiert wird. Das Named-Pipe-Dateisystem bildet das Pseudo-Geräte-Objekt `NamedPipe` aus. Dieses ist im Gerätestapel des IO-Managers enthalten. Um die Kommunikation zwischen Prozessen und Serviceanfragen von Anwendungsprozessen an Dienstprozessen zu erkennen, verwendet Sysmon in etwa die gleiche Filter-Treiber-Funktionalität, wie für die Erkennung von klassischen Dateizugriffen. Dabei erstellt das Sysmon-Kernel-Modul (`SysmonDrv.sys`) ein neues Geräte-Objekt beim IO-Manager mit dem Namen `SysmonPipeFilter` (`\Device\SysmonPipeFilter`) und fügt es dem Gerätestapel hinzu, sodass es oberhalb des Named-Pipe-Dateisystem-Gerätes liegt und somit bei einer Anfrage an das Named-Pipe-System vorher durchlaufen werden muss. Der Sysmon-Treiber (`Sysmon.sys`) registriert eigene Routinen für die Filterung in der IRP-Tabelle des Geräteobjektes, was dazu führt, dass Objekt-Anfragen bezüglich Named-Pipes durch den Filter laufen und zum Aufruf der entsprechenden Routinen im Sysmon-Treiber führen. Zugriffe auf Named-Pipe-Objekte können somit vom Sysmon-Treiber erkannt werden und der Dienstanwendung gemeldet werden [LB19, S. 24ff][Mon22].

4.3.7 Erkennung von Netzwerk-Ereignissen

Um Netzwerk-Ereignisse zu erkennen verwendet Sysmon Event-Tracing-For-Windows (ETW). Dafür erstellt und startet die User-Mode-Dienstanwendung eine ETW-System-Session im ETW-System unter dem Namen „`SYSMON TRACE`“. In der System-Session konfiguriert der Sysmon-Dienst, dass Event-Objekte der System-Provider-Klasse „NT Kernel Trace“ gesammelt werden sollen. Zusätzlich wird ein Keyword-Filter konfiguriert, sodass nur Netzwerk-Ereignisse in der Session gesammelt werden. Dafür setzt der Sysmon-Dienst in den Enable-Flags der System-Provider-Klasse das Flag `EVENT_TRACE_FLAG_NETWORK_TCPIP`. Die Session wird zudem als Real-Time-Session eingestellt, sodass Sysmon als Consumer die eingehenden Event-Objekte live entgegennehmen kann [LB19, S. 32ff][Mon22].

Der Sysmon-User-Mode-Dienst agiert gleichzeitig als ETW-Consumer und lässt sich benachrichtigen, wenn neue Event-Objekte in der Session bereitstehen. Die bezogenen Event-Daten eines Event-Objektes sind nach dem Erhalt vom ETW-System serialisiert und speziell codiert. Die ETW-Anwendung nimmt das Dekodieren selbst vor. Dafür durchsucht der Sysmon-Prozess das WMI-Repo (`root\wmi`) nach den notwendigen MOF-Klassen, die die Event-Objekttypen der Netzwerkereignisse des „NT Kernel Trace“-Providers beschreiben. Diese sind notwendig, um die Event-Daten korrekt zu deserialisieren [LB19, S. 32ff][Mon22].

4.3.8 Erkennung von WMI-Operationen

Zur Erkennung von sicherheitsrelevanten WMI-Vorgängen registriert der Sysmon-Dienst eine Callback-Funktion in der WMI-Bibliothek, die vom WMI-Dienst alle 5 Sekunden aufgerufen wird. Mit der Callback werden Anfragen auf die drei WMI-Klassen `__EventConsumer`, `__EventFilter` und `__FilterToConsumerBinding` gefiltert. Eine dieser drei Klassen ist immer involviert, wenn persistierte Änderungen am WMI-System vorgenommen werden. Dies sind die sicherheitsrelevanten Vorgänge, die erkannt werden sollen. Sysmon versucht nur Änderungen am WMI-System zu erkennen, die auch persistent sind. Sysmon prüft innerhalb der Callback-Routine, ob eine der drei Aktionen `Deletion`, `Creation` oder `Modification` der drei Klassen angewendet wurde. Wird eine solche WMI-Anfrage erkannt, wird ein entsprechendes Sysmon-Event-Objekt erzeugt [LB19, S. 28ff][Mon22].

4.4 Sysmon-Ereignistypen

Prozess-Ereignisse

1	Process Created
5	Process Terminated

Abbild-Lade-Ereignisse

7	Image Loaded

Datei-Ereignisse

11	File Created Or Modified
15	File Alternate Data Stream Created Or Modified
26	File Deleted
23	File Deleted (archived)
2	File Creation Time Changed

Registry-Ereignisse

12	Registry Object Added Or Deleted
13	Registry Value Set
14	Registry Object Renamed

Netzwerk-Ereignisse

3	Network Connection Detected
22	DNS Query Detected

Pipe-Ereignisse (Interprozesskommunikation)

17	Pipe Created
18	Pipe Connected

Zwischenablage-Ereignisse

24	Clipboard Changed

WMI-Ereignisse

19	WMI Filter Activity Detected
20	WMI Consumer Activity Detected
21	WMI Binding Activity Detected

Zugriff auf fremde Prozesse

8	Remote Thread Creation Detected
10	Remote Process Access Detected
25	Process Tampering Detected

Direkter Zugriff auf Datenträger

9	Raw Access Read Detected

Kernel-Treiber-Ereignisse

6	Driver Loaded

Sysmon-Ereignisse

4	Service State Changed
16	Configuration Changed

Tabelle 4.2: Übersicht Sysmon-Ereignistypen (Version: 13) [Har21a][Mon22]

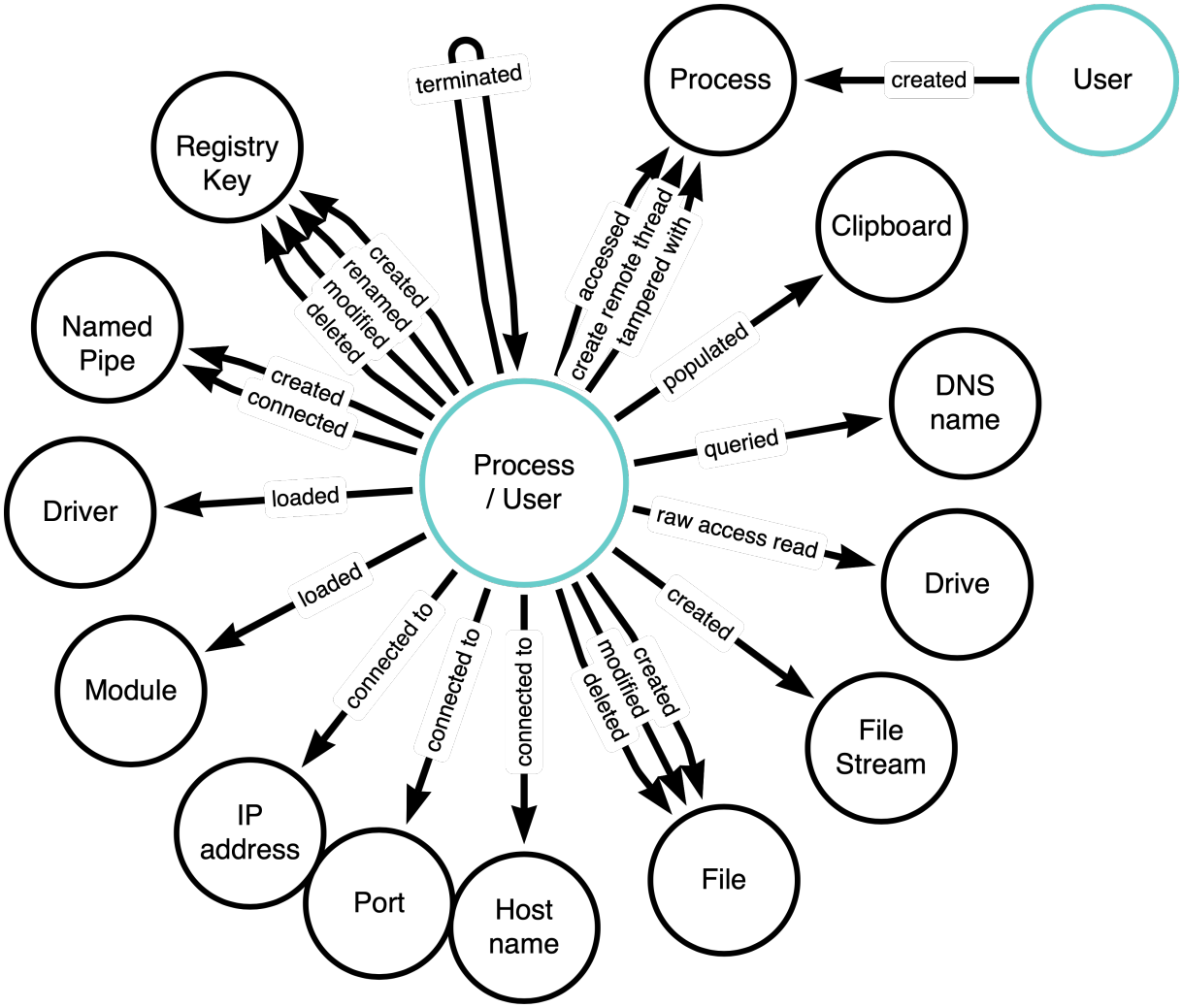

Abbildung 4.2: Sysmon-Event-Informationen [Har21b]

4.4.1 Prozess-Ereignisse

Process Created (ID: 1)

Attribut	Datentyp	Beschreibung
RuleName	UnicodeString	konfigurierte Sysmon-Regel, die mit diesem Ereignis übereinstimmt
UtcTime	UincodeString	Auftrittszeitpunkt des Ereignisses
ParentProcessGuid	GUID	GUID des Elternprozesses
ParentProcessId	UInt32	ID des Elternprozesses
ParentImage	UnicodeString	Dateipfad des Images des Elternprozesses
ParentCommandLine	UincodeString	Kommandozeile des Elterprozesses
ParentUser	UincodeString	Benutzerkonto, unter dem der Elternprozess ausgeführt wird
ProcessGuid	GUID	GUID des neu erzeugten Prozesses
ProcessId	UInt32	ID des neu erzeugten Prozesses
Image	UnicodeString	Dateipfad des Executables des neu erzeugten Prozesses
CommandLine	UnicodeString	Kommandozeile des neu erzeugten Prozesses
FileVersion	UnicodeString	Versionsangabe aus Metadaten des Executables des neu erzeugten Prozesses
Description	UnicodeString	Beschreibung aus Metadaten des Executables des neu erzeugten Prozesses
Product	UnicodeString	Produktbeschreibung aus Metadaten des Executables des neu erzeugten Prozesses
Company	UnicodeString	Firmenname aus Metadaten des Executables des neu erzeugten Prozesses
OriginalFileName	UnicodeString	Original-Dateiname aus Metadaten des Executables des neu erzeugten Prozesses
Hashes	UnicodeString	Prüfsumme(n) des Executables des neu erzeugten Prozesses
CurrentDirectory	UnicodeString	Arbeitsverzeichnis, in dem der neu erzeugte Prozess ausgeführt wird
User	UnicodeString	Benutzerkonto, unter dem der neu erzeugte Prozess ausgeführt wird
LogonGuid	GUID	GUID der Anmeldesitzung, in der der neu erzeugten Prozess ausgeführt wird
LogonId	HexInt64	ID der Anmeldesitzung, in der der neu erzeugten Prozess ausgeführt wird
TerminalSessionId	UInt32	ID der Terminal-Sitzung, in der der neu erzeugten Prozesses ausgeführt wird
IntegrityLevel	UincodeString	MIC-Integrity-Level (low, medium, high, system) des neu erzeugten Prozesses

Tabelle 4.3: Sysmon Process Created (Version: 13) [Har21a][Mon22]

Process Terminated (ID: 5)

Attribut	Datentyp	Beschreibung
RuleName	UincodeString	konfigurierte Sysmon-Regel, die mit diesem Ereignis übereinstimmt
UtcTime	UincodeString	Auftrittszeitpunkt des Ereignisses
ProcessGuid	UincodeString	GUID des Prozesses, der sich beendet hat
ProcessId	UInt32	ID des Prozesses, der sich beendet hat
Image	UnicodeString	Dateipfad des Executables des Prozesses, der sich beendet hat
User	UnicodeString	Benutzerkonto, unter dem der Prozess ausgeführt wurde, der sich beendet hat

Tabelle 4.4: Sysmon Process Terminated (Version: 13) [Har21a][Mon22]

4.4.2 Abbild-Lade-Ereignisse

Image Loaded (ID: 7)

Attribut	Datentyp	Beschreibung
RuleName	UnicodeString	konfigurierte Sysmon-Regel, die mit diesem Ereignis übereinstimmt
UtcTime	UnicodeString	Auftrittszeitpunkt des Ereignisses
ProcessGuid	GUID	GUID des Prozesses, der das Image geladen hat
ProcessId	UInt32	ID des Prozesses, der das Image geladen hat
Image	UnicodeString	Dateipfad des Executables des Prozesses, der das Image geladen hat
User	UnicodeString	Benutzerkonto, unter dem der Prozess ausgeführt wird, der das Image geladen hat
ImageLoaded	UnicodeString	Dateipfad des geladenen Images
FileVersion	UnicodeString	Versionsangabe aus Metadaten des geladenen Images
Description	UnicodeString	Beschreibung aus Metadaten des geladenen Images
Product	UnicodeString	Produktbezeichnung aus Metadaten des geladenen Images
Company	UnicodeString	Firmenname aus Metadaten des geladenen Images
OriginalFileName	UnicodeString	Original-Dateiname aus Metadaten des geladenen Images
Hashes	UnicodeString	Prüfsummen des geladenen Images
Signed	UnicodeString	Status, ob das Image signiert ist
Signature	UnicodeString	Bezeichnung der Stelle, die das geladene Image signiert hat
SignatureStatus	UnicodeString	Status der Signatur des geladenen Images

Tabelle 4.5: Sysmon Image Loaded (Version: 13) [Har21a][Mon22]

4.4.3 Datei-Ereignisse

File Created Or Modified (ID: 11)

Attribut	Datentyp	Beschreibung
RuleName	UnicodeString	konfigurierte Sysmon-Regel, die mit diesem Ereignis übereinstimmt
UtcTime	UnicodeString	Auftrittszeitpunkt des Ereignisses
ProcessGuid	GUID	GUID des Prozesses, der die Datei erstellt oder bearbeitet hat
ProcessId	UInt32	ID des Prozesses, der die Datei erstellt oder bearbeitet hat
Image	UnicodeString	Dateipfad des Executables des Prozesses, der die Datei erstellt oder bearbeitet hat
User	UnicodeString	Benutzerkonto, unter dem der Prozesses ausgeführt wird, der die Datei erstellt oder bearbeitet hat
TargetFilename	UnicodeString	Dateipfad der Datei, die neu erstellt oder bearbeitet wurde
CreationUtcTime	UnicodeString	Zeitpunkt, zudem die Datei erstellt oder bearbeitet wurde

Tabelle 4.6: Sysmon File Created Or Modified (Version: 13) [Har21a][Mon22]

File Alternate Data Stream Created Or Modified (ID: 15)

Attribut	Datentyp	Beschreibung
RuleName	UnicodeString	konfigurierte Sysmon-Regel, die mit diesem Ereignis übereinstimmt
UtcTime	UnicodeString	Auftrittszeitpunkt des Ereignisses
ProcessGuid	GUID	GUID des Prozesses, der den ADS erstellt oder bearbeitet hat
ProcessId	UInt32	ID des Prozesses, der den ADS erstellt oder bearbeitet hat
Image	UnicodeString	Dateipfad des Executables des Prozesses, der den ADS erstellt oder bearbeitet hat
User	UnicodeString	Benutzerkonto, unter dem der Prozess ausgeführt wird, der den ADS erstellt oder bearbeitet hat
TargetFilename	UnicodeString	Dateipfad der Datei, in dem ein neuer ADS erstellt oder ein bestehender ADS bearbeitet wurde
CreationUtcTime	UnicodeString	Zeitpunkt, zudem der ADS erstellt oder bearbeitet wurde
Hash	UnicodeString	Prüfsumme der gesamten geänderten Datei
Contents	UnicodeString	Inhalt der in den ADS geschrieben wurde

Tabelle 4.7: Sysmon File Alternate Data Stream Created Or Modified (Version: 13) [Har21a][Mon22]

File Deleted (ID: 26)

Attribut	Datentyp	Beschreibung
RuleName	UnicodeString	konfigurierte Sysmon-Regel, die mit diesem Ereignis übereinstimmt
UtcTime	UnicodeString	Auftrittszeitpunkt des Ereignisses
ProcessGuid	GUID	GUID des Prozesses, der die Datei gelöscht hat
ProcessId	UInt32	ID des Prozesses, der die Datei gelöscht hat
Image	UnicodeString	Dateipfad des Executables des Prozesses, der die Datei gelöscht hat
User	UnicodeString	Benutzerkonto, unter dem der Prozess ausgeführt wurde, der die Datei gelöscht hat
TargetFilename	UnicodeString	Dateipfad der Datei die gelöscht wurde
Hashes	UnicodeString	Prüfsumme der Datei die gelöscht wurde
IsExecutable	Boolean	Gibt an, ob die gelöschte Datei eine PE-Datei war

Tabelle 4.8: Sysmon File Deleted (Version: 13) [Har21a][Mon22]

File Deleted (archived) (ID: 23)

Attribut	Datentyp	Beschreibung
RuleName	UnicodeString	konfigurierte Sysmon-Regel, die mit diesem Ereignis übereinstimmt
UtcTime	UnicodeString	Auftrittszeitpunkt des Ereignisses
ProcessGuid	GUID	GUID des Prozesses, der die Datei gelöscht hat
ProcessId	UInt32	ID des Prozesses, der die Datei gelöscht hat
Image	UnicodeString	Dateipfad des Executables des Prozesses, der die Datei gelöscht hat
User	UnicodeString	Benutzerkonto, unter dem der Prozess ausgeführt wurde, der die Datei gelöscht hat
TargetFilename	UnicodeString	Dateipfad der Datei die gelöscht wurde
Hashes	UnicodeString	Prüfsumme der Datei die gelöscht wurde
IsExecutable	Boolean	Gibt an, ob die gelöschte Datei eine PE-Datei war
Archived	UnicodeString	Gibt an, ob die gelöschte Datei vor dem Löschen durch Sysmon im Archiv-Ordner gesichert wurde

Tabelle 4.9: Sysmon File Deleted (archived) (Version: 13) [Har21a][Mon22]

File Creation Time Canged (ID: 2)

Attribut	Datentyp	Beschreibung
RuleName	UnicodeString	konfigurierte Sysmon-Regel, die mit diesem Ereignis übereinstimmt
UtcTime	UnicodeString	Auftrittszeitpunkt des Ereignisses
ProcessGuid	GUID	GUID des Prozesses, der das Erstellungszeit-Attribut der Datei geändert hat
ProcessId	UInt32	ID des Prozesses, der das Erstellungszeit-Attribut der Datei geändert hat
Image	UnicodeString	Dateipfad des Executables des Prozesses, der das Erstellungszeit-Attribut der Datei geändert hat
User	UnicodeString	Benutzerkonto, unter dem der Prozesses ausgeführt wird, der das Erstellungszeit-Attribut der Datei geändert hat
TargetFilename	UnicodeString	Dateipfad der Datei, dessen Erstellungszeit-Attribut geändert wurde
CreationUtcTime	UnicodeString	neuer Zeitpunkt, der als Erstellungszeitpunkt hinterlegt wurde
PreviousCreationUtcTime	UnicodeString	vorheriger Erstellungszeitpunkt, der überschrieben wurde

Tabelle 4.10: Sysmon File Creation Time Changed (Version: 13) [Har21a][Mon22]

4.4.4 Registry-Ereignisse

Registry Object Added Or Deleted (ID: 12)

Attribut	Datentyp	Beschreibung
RuleName	UnicodeString	konfigurierte Sysmon-Regel, die mit diesem Ereignis übereinstimmt
EventType	UnicodeString	Aktionstyp (Schlüssel erstellt, Schlüssel gelöscht, Wert erstellt, Wert gelöscht)
UtcTime	UnicodeString	Auftrittszeitpunkt des Ereignisses
ProcessGuid	GUID	GUID des Prozesses, der das Registry-Objekt erstellt oder gelöscht hat
ProcessId	UInt32	ID des Prozesses, der das Registry-Objekt erstellt oder gelöscht hat
Image	UnicodeString	Dateipfad des Executables des Prozesses, der das Registry-Objekt erstellt oder gelöscht hat
User	UnicodeString	Benutzerkonto, unter dem der Prozesses ausgeführt wird, der das Registry-Objekt erstellt oder gelöscht hat
TargetObject	UnicodeString	Registry-Pfad des Schlüssels oder Wertes, der erstellt oder gelöscht wurde

Tabelle 4.11: Sysmon Registry Object Added or Deleted (Version: 13) [Har21a][Mon22]

Registry Value Set (ID: 13)

Attribut	Datentyp	Beschreibung
RuleName	UnicodeString	konfigurierte Sysmon-Regel, die mit diesem Ereignis übereinstimmt
EventType	UnicodeString	Aktionstyp (Wert geändert)
UtcTime	UnicodeString	Auftrittszeitpunkt des Ereignisses
ProcessGuid	GUID	GUID des Prozesses, der den Registry-Wert geändert hat
ProcessId	UInt32	ID des Prozesses, der den Registry-Wert geändert hat
Image	UnicodeString	Dateipfad des Executables des Prozesses, der den Registry-Wert geändert hat
User	UnicodeString	Benutzerkonto, unter dem der Prozesses ausgeführt wird, der den Registry-Wert geändert hat
TargetObject	UnicodeString	Registry-Pfad des Schlüssels, dessen Werte-Feld geändert wurde
Details	UnicodeString	Bezeichnung des Werte-Feldes, das geändert wurde

Tabelle 4.12: Sysmon Registry Value Set (Version: 13) [Har21a][Mon22]

Registry Object Renamed (ID: 14)

Attribut	Datentyp	Beschreibung
RuleName	UnicodeString	konfigurierte Sysmon-Regel, die mit diesem Ereignis übereinstimmt
EventType	UnicodeString	Aktionstyp (Schlüssel umbenannt, Wert umbenannt)
UtcTime	UnicodeString	Auftrittszeitpunkt des Ereignisses
ProcessGuid	GUID	GUID des Prozesses, der das Registry-Objekt umbenannt hat
ProcessId	UInt32	ID des Prozesses, der das Registry-Objekt umbenannt hat
Image	UnicodeString	Dateipfad des Executables des Prozesses, der das Registry-Objekt umbenannt hat
User	UnicodeString	Benutzerkonto unter, dem der Prozesses ausgeführt wird, der das Registry-Objekt umbenannt hat
TargetObject	UnicodeString	Registry-Pfad des Registry-Objektes das umbenannt wurde (vorheriger Name)
NewName	UnicodeString	neuer Name in den das Registry-Objekt umbenannt wurde

Tabelle 4.13: Sysmon Registry Object Renamed (Version: 13) [Har21a][Mon22]

4.4.5 Netzwerk-Ereignisse

Network Connection Detected (ID: 3)

Attribut	Datentyp	Beschreibung
RuleName	UnicodeString	konfigurierte Sysmon-Regel, die mit diesem Ereignis übereinstimmt
UtcTime	UnicodeString	Auftrittszeitpunkt des Ereignisses
ProcessGUID	GUID	GUID des Prozesses, der die Netzwerkverbindung aufgebaut hat
ProcessId	UInt32	ID des Prozesses, der die Netzwerkverbindung aufgebaut hat
Image	UnicodeString	Dateipfad des Executables des Prozesses, der die Netzwerkverbindung aufgebaut hat
User	UnicodeString	Benutzerkonto, unter dem der Prozesses ausgeführt wird, der die Netzwerkverbindung aufgebaut hat
Protocol	UnicodeString	gibt das benutzte Transport-Protokoll an (TCP, UDP)
Initiated	Boolean	gibt an, ob die Verbindung von diesem Prozess gestartet wurde
SourceIsIpv6	Boolean	gibt an, ob die Quelladresse eine IPv6-Adresse ist
SourceIp	UnicodeString	Quell-IP-Adresse
SourceHostname	UnicodeString	Quell-Hostname (mittels Reverse-DNS durch Sysmon ermittelt)
SourcePort	UInt16	Quell-Port-Nummer
SourcePortName	UnicodeString	Protokoll, das mit der Quell-Port-Nummer assoziiert wird
DestinationIsIpv6	Boolean	gibt an, ob die Ziel-IP-Adresse eine IPv6-Adresse ist
DestinationIp	UnicodeString	Ziel-IP-Adresse
DestinationHostname	UnicodeString	Ziel-Hostname (mittels Reverse-DNS durch Sysmon ermittelt)
DestinationPort	UInt16	Ziel-Port-Nummer
DestinationPortName	UnicodeString	Protokoll, das mit der Ziel-Port-Nummer assoziiert wird

Tabelle 4.14: Sysmon Network Connection Detected (Version: 13) [Har21a][Mon22]

DNS Query Detected (ID: 22)

Attribut	Datentyp	Beschreibung
RuleName	UnicodeString	konfigurierte Sysmon-Regel, die mit diesem Ereignis übereinstimmt
UtcTime	UnicodeString	Auftrittszeitpunkt des Ereignisses
ProcessGUID	GUID	GUID des Prozesses, der die Namensauflösung angefragt hat
ProcessId	UInt32	ID des Prozesses, der die Namensauflösung angefragt hat
Image	UnicodeString	Dateipfad des Executables des Prozesses, der die Namensauflösung angefragt hat
User	UnicodeString	Benutzerkonto, unter dem der Prozesses ausgeführt wird, der die Namensauflösung angefragt hat
QueryName	UnicodeString	DNS-Name der angefragt wurde
QueryStatus	UnicodeString	Ergebnis-Status-Code der Namensauflösung
QueryResults	UnicodeString	Ergebnis der Namensauflösung

Tabelle 4.15: Sysmon DNS Query Detected (Version: 13) [Har21a][Mon22]

4.4.6 Pipe-Ereignisse (Interprozesskommunikation)

Pipe Created (ID: 17)

Attribut	Datentyp	Beschreibung
RuleName	UnicodeString	konfigurierte Sysmon-Regel, die mit diesem Ereignis übereinstimmt
EventType	UnicodeString	Aktionstyp (Named-Pipe erstellt)
UtcTime	UnicodeString	Auftrittszeitpunkt des Ereignisses
ProcessGUID	GUID	GUID des Prozesses, der die Named-Pipe erstellt hat
ProcessId	UInt32	ID des Prozesses, der die Named-Pipe erstellt hat
Image	UnicodeString	Dateipfad des Executables des Prozesses, der die Named-Pipe erstellt hat
User	UnicodeString	Benutzerkonto, unter dem der Prozess läuft, der die Named-Pipe erstellt hat
PipeName	UnicodeString	Objekt-Pfad des neu erstellten Named-Pipe-Objektes

Tabelle 4.16: Sysmon Pipe Created (Version: 13) [Har21a][Mon22]

Pipe Connected (ID: 18)

Attribut	Datentyp	Beschreibung
RuleName	UnicodeString	konfigurierte Sysmon-Regel, die mit diesem Ereignis übereinstimmt
EventType	UnicodeString	Aktionstyp (Named-Pipe verbunden)
UtcTime	UnicodeString	Auftrittszeitpunkt des Ereignisses
ProcessGUID	GUID	GUID des Prozesses, der sich mit der Named-Pipe verbunden hat (das Named-Pipe-Objekt geöffnet hat)
ProcessId	UInt32	ID des Prozesses, der sich mit der Named-Pipe verbunden hat (das Named-Pipe-Objekt geöffnet hat)
Image	UnicodeString	Dateipfad des Executables des Prozesses, der sich mit der Named-Pipe verbunden hat (das Named-Pipe-Objekt geöffnet hat)
User	UnicodeString	Benutzerkonto, unter dem der Prozesses ausgeführt wird, der sich mit der Named-Pipe verbunden hat (das Named-Pipe-Objekt geöffnet hat)
PipeName	UnicodeString	Objekt-Pfad des Named-Pipe-Objektes, mit dem sich verbunden wurde (das geöffnet wurde)

Tabelle 4.17: Sysmon Pipe Connected (Version: 13) [Har21a][Mon22]

4.4.7 WMI-Ereignisse

WMI Filter Activity Detected (ID: 19)

Attribut	Datentyp	Beschreibung
RuleName	UnicodeString	konfigurierte Sysmon-Regel, die mit diesem Ereignis übereinstimmt
EventType	UnicodeString	Aktionstyp (WmiFilterEvent)
UtcTime	UnicodeString	Auftritszeitpunkt des Ereignisses
Query	UnicodeString	Anfrage, welche die WMI-Operation ausgelöst hat
Operation	UnicodeString	Operationstyp (Created, Modified, Deleted)
User	UnicodeString	Benutzerkonto, unter dem die WMI-Operation durchgeführt wurde
Name	UnicodeString	Name des WMI-Filters der erstellt, gelöscht oder geändert wurde
EventNamespace	UnicodeString	Event-Namespace des WMI-Filters der erstellt, gelöscht oder geändert wurde

Tabelle 4.18: Sysmon WMI Filter Activity (Version: 13) [Har21a][Mon22]

WMI Consumer Activity Detected (ID: 20)

Attribut	Datentyp	Beschreibung
RuleName	UnicodeString	konfigurierte Sysmon-Regel, die mit diesem Ereignis übereinstimmt
EventType	UnicodeString	Aktionstyp (WmiConsumerEvent)
UtcTime	UnicodeString	Auftrittszeitpunkt des Ereignisses
Operation	UnicodeString	Operationstyp (Created, Modified, Deleted)
User	UnicodeString	Benutzerkonto, unter dem die WMI-Operation durchgeführt wurde
Name	UnicodeString	Name des WMI-Consumers der erstellt, gelöscht oder geändert wurde
Type	UnicodeString	Typ des WMI-Consumers der erstellt, gelöscht oder geändert wurde
Destination	UnicodeString	Dateipfad der Datensenke des WMI-Consumers der erstellt, gelöscht oder geändert wird

Tabelle 4.19: Sysmon WMI Consumer Activity Detected (Version: 13) [Har21a][Mon22]

WMI Binding Activity Detected (ID: 21)

Attribut	Datentyp	Beschreibung
RuleName	UnicodeString	konfigurierte Sysmon-Regel, die mit diesem Ereignis übereinstimmt
EventType	UnicodeString	Aktionstyp (WmiBindingEvent)
UtcTime	UnicodeString	Auftrittszeitpunkt des Ereignisses
Operation	UnicodeString	Operationstyp (Created, Modified, Deleted)
User	UnicodeString	Benutzerkonto, unter dem die WMI-Operation durchgeführt wurde
Consumer	UnicodeString	WMI-Consumer, des WMI-Binding-Objektes, das erstellt, gelöscht oder geändert wurde
Filter	UnicodeString	WMI-Filter, des WMI-Bining-Objektes, das erstellt, gelöscht oder geändert wurde

Tabelle 4.20: Sysmon WMI Binding Activity Detected (Version: 13) [Har21a][Mon22]

4.4.8 Zwischenablage-Ereignisse

Clipboard Changed (ID: 24)

Attribut	Datentyp	Beschreibung
RuleName	UnicodeString	konfigurierte Sysmon-Regel, die mit diesem Ereignis übereinstimmt
UtcTime	UnicodeString	Auftrittszeitpunkt des Ereignisses
ProcessGUID	GUID	GUID des Processes, der Daten in die Zwischenablage gespeichert hat
ProcessId	UInt32	ID des Processes, der Daten in die Zwischenablage gespeichert hat
Image	UnicodeString	Dateipfad des Executables des Processes, der Daten in die Zwischenablage gespeichert hat
User	UnicodeString	Benutzerkonto, unter dem der Process ausgeführt wird, der Daten in die Zwischenablage gespeichert hat
Session	UInt32	Terminal-Session der die Zwischenablage zugeordnet ist
ClientInfo	UnicodeString	Benutzerkonto und Hostname des assoziierten Remote-Desktop-Session-Host (falls es sich um eine Remote-Desktop-Sessio handelt)
Hashes	UnicodeString	Prüfsumme(n) der Daten, die in die Zwischenablage geschrieben wurden
Archived	Boolean	gibt an, ob die Daten, die in die Zwischenablage geschrieben wurden, durch Sysmon im Archiv-Ordner gesichert wurden

Tabelle 4.21: Sysmon Clipboard Changed (Version: 13) [Har21a][Mon22]

4.4.9 Zugriffsereignisse auf fremde Prozesse

Remote Thread Creation Detected (ID: 8)

Attribut	Datentyp	Beschreibung
RuleName	UnicodeString	konfigurierte Sysmon-Regel, die mit diesem Ereignis übereinstimmt
UtcTime	UnicodeString	Auftrittszeitpunkt des Ereignisses
SourceProcessGUID	GUID	GUID des Prozesses, der den Remote-Thread erstellt hat
SourceProcessId	UInt32	ID des Prozesses, der den Remote-Thread erstellt hat
SourceImage	UnicodeString	Dateipfad des Executables des Prozesses, der den Remote-Thread erstellt hat
SourceUser	UnicodeString	Benutzerkonto, unter dem der Prozess ausgeführt wird, der den Remote-Thread erstellt hat
TargetProcessGUID	GUID	GUID des Prozesses, dem der neu erstellte Remote-Thread angehört
TargetProcessId	UInt32	ID des Prozesses, dem der neu erstellte Remote-Thread angehört
TargetImage	UnicodeString	Dateipfad des Executables des Prozesses, dem der neu erstellte Remote-Thread angehört
TargetUser	UnicodeString	Benutzerkonto, unter dem der Prozess ausgeführt wird, dem der neu erstellte Remote-Thread angehört
NewThreadId	UInt32	ID des neu erstellten Threads
StartAddress	UincodeString	Start-Adresse an der die Ausführung des neu erstellten Threads beginnt
StartModule	UincodeString	Name des geladenen Moduls, in dem die Ausführung des neu erstellten Threads beginnt (von Sysmon aus der Loaded-Module-Liste des PEB ermittelt)
StartFunction	UincodeString	Name der exportierte Funktion, an dessen Anfang die Ausführung des neu erstellten Threads beginnt (von Sysmon aus der Export-Tabelle des Start-Moduls ermittelt; funktioniert nur, wenn Adresse genau übereinstimmt)

Tabelle 4.22: Sysmon Remote Thread Creation Detected (Version: 13) [Har21a][Mon22]

Remote Process Access Detected (ID: 10)

Attribut	Datentyp	Beschreibung
RuleName	UincodeString	konfigurierte Sysmon-Regel, die mit diesem Ereignis übereinstimmt
UtcTime	UincodeString	Auftrittszeitpunkt des Ereignisses
SourceProcessGUID	GUID	GUID des Prozesses, der auf den fremden Prozess zugegriffen hat
SourceProcessId	Uint32	ID des Prozesses, der auf den fremden Prozess zugegriffen hat
SourceThreadId	UInt32	ID des Threads, der auf den fremden Prozess zugegriffen hat
SourceImage	UnicodeString	Dateipfad des Executable des Prozesses, der auf den fremden Prozess zugegriffen hat
SourceUser	UnicodeString	Benutzerkonto, unter dem der Prozess ausgeführt wird, der auf den fremden Prozess zugegriffen hat
TargetProcessGUID	GUID	GUID des Prozesses auf den zugegriffen wurde
TargetProcessId	UInt32	ID des Prozesses auf den zugegriffen wurde
TargetImage	UnicodeString	Dateipfad des Executable des Prozesses, auf den zugegriffen wurde
TargetUser	UnicodeString	Benutzerkonto, unter dem der Prozess ausgeführt wird, auf den zugegriffen wurde
GrantedAccess	HexInt32	Access-Flags (Bitmaske) des Zugriffs
CallTrace	UnicodeString	Stack-Trace des Zeitpunktes, an dem die API-Funktion OpenProcess aufgerufen wurde

Tabelle 4.23: Sysmon Remote Process Access Detected (Version: 13) [Har21a][Mon22]

Process Tampering Detected (ID: 25)

Attribut	Datentyp	Beschreibung
RuleName	UnicodeString	konfigurierte Sysmon-Regel, die mit diesem Ereignis übereinstimmt
UtcTime	UnicodeString	Auftrittszeitpunkt des Ereignisses
ProcessGUID	GUID	GUID des Prozesses, der manipuliert wurde
ProcessId	UInt32	ID des Prozesses, der manipuliert wurde
Image	UnicodeString	Dateipfad des Executables des Prozesses, der manipuliert wurde
User	UnicodeString	Benutzerkonto, unter dem der Prozess ausgeführt wird, der manipuliert wurde
Type	UincodeString	Art der Manipulation (z.B. Prozess-Hollowing, Process-Herpaderping, Image is replaced)

Tabelle 4.24: Sysmon Process Tampering Detected (Version: 13) [Har21a][Mon22]

4.4.10 Zugriffsereignisse auf Datenträger

Raw Access Read Detected (ID: 9)

Attribut	Datentyp	Beschreibung
RuleName	UnicodeString	konfigurierte Sysmon-Regel, die mit diesem Ereignis übereinstimmt
UtcTime	UnicodeString	Auftrittszeitpunkt des Ereignisses
ProcessGuid	GUID	GUID des Prozesses, der am Dateisystem vorbei direkt auf die Daten des Datenträgers zugegriffen hat
ProcessId	UInt32	ID des Prozesses, der am Dateisystem vorbei direkt auf die Daten des Datenträgers zugegriffen hat
Image	UnicodeString	Dateipfad des Executables des Prozesses, der am Dateisystem vorbei direkt auf die Daten des Datenträgers zugegriffen hat
User	UnicodeString	Benutzerkonto, unter dem der Prozess ausgeführt wird, der am Dateisystem vorbei direkt auf die Daten des Datenträgers zugegriffen hat
Device	UnicodeString	Geräte-Objekt des Datenträgers auf den zugegriffen wurde

Tabelle 4.25: Sysmon Raw Access Read Detected (Version: 13) [Har21a][Mon22]

4.4.11 Kernel-Treiber-Ereignisse

Driver Loaded (ID: 6)

Attribut	Datentyp	Beschreibung
RuleName	UnicodeString	konfigurierte Sysmon-Regel, die mit diesem Ereignis übereinstimmt
UtcTime	UnicodeString	Auftrittszeitpunkt des Ereignisses
ImageLoaded	UnicodeString	Dateipfad des Executable des geladenen Kernel-Treibers
Hashes	UnicodeString	Prüfsumme(n) des geladenen Kernel-Moduls
Signed	UnicodeString	gibt an, ob der geladene Kernel-Treiber signiert ist
Signature	UnicodeString	Signatur-Informationen
SignatureStatus	UnicodeString	Signatur-Status

Tabelle 4.26: Sysmon Driver Loaded (Version: 13) [Har21a][Mon22]

4.4.12 Sysmon-Ereignisse

Service State Changed (ID: 4)

Attribut	Datentyp	Beschreibung
UtcTime	UnicodeString	Auftrittszeitpunkt des Ereignisses
State	UnicodeString	Status der Sysmon-Dienstanwendung (gestartet, angehalten)
Version	UnicodeString	Version der Sysmon-Software
SchemaVersion	UnicodeString	Unterstützte Version des XML-Schema, auf der die Konfigurationsdatei basieren darf (abwärtskompatibel)

Tabelle 4.27: Sysmon Service State Changed (Version: 13) [Har21a][Mon22]

Configuration Changed (ID: 16)

Attribut	Datentyp	Beschreibung
UtcTime	UnicodeString	Auftrittszeitpunkt des Ereignisses
Configuration	UnicodeString	Dateipfad der neu geladenen Konfigurationsdatei
ConfigurationFileHash	UnicodeString	Prüfsumme(n) der neu geladenen Konfigurationsdatei

Tabelle 4.28: Sysmon Configuration Changed (Version: 13) [Har21a][Mon22]

4.5 Einschätzung bezüglich der Erkennung

Sysmon überwacht die meisten relevanten Aktionen, die von Malware ausgeführt werden. Leider ermöglicht Windows es, viele Aktionen neben dem offiziell vorgesehenen Standardweg auch sehr untypisch in einer anderen Weise auszuführen. Dies wird von Malware zum Teil genutzt, um Erkennungsmechanismen zu umgehen. Sysmon erkennt in diesen Fällen nicht alle Wege. Ein blinder Fleck in der Erkennung liegt über den Aktionen, in die der Win32-Kernel (`win32k.sys`) involviert ist. Dies betrifft die von Microsoft als „Desktop-Technologien“ bezeichneten Funktionen (Benutzeroberfläche, Dialoge, GDI, Steuerelemente-Funktionen, COM, OLE usw.). Diese werden von Sysmon nicht überwacht. Eine Malware könnte sich vor dem Sysmon-Monitoring vermutlich weitestgehend verstecken, wenn diese eine Menge Tricks anwendet. Üblicherweise nutzt Malware jedoch nur selten diese speziellen Tricks, die von Sysmon nicht erkannt werden für einige wenige Aktionen. Überwiegend werden die Standard-Verfahren oder nicht-Standard-Varianten dieser Verfahren genutzt, die von Sysmon erkannt werden [LB19, S. 35].

Ein weiteres Problem bei der Erkennung an dieser Stelle ist, dass viele Firmen dieselben öffentlich geteilten Sysmon-Konfigurationen nutzen (Sysmon-Config von SwiftOnSecurity). Diese Konfigurationen gelten als gut, da sie von vielen eingesetzt werden und eine

gute handhabbare Menge an Ereignissen liefern, die sich zentral gesammelt gut weiterverarbeiten lässt. Dabei ist es den Firmen wichtig, nicht zu viele nicht relevante Ereignisse zu sammeln, damit die Verarbeitung handhabbar bleibt. Daher muss in jedem Fall eines produktiven Einsatzes in Kombination mit einem SIEM eine praktische Vor-Filterung in Sysmon geschehen. Angreifer kennen die allgemein eingestellten Filterregeln der gängigen Sysmon-Konfigurationen, die in Firmen eingesetzten werden und können dieses Informationsleck nutzen, indem sie versuchen das Verhalten ihrer Malware so zu gestalten, dass möglichst nur solche Ereignisse erkannt werden, für die aufgrund der Filterregen keine Sysmon-Event-Objekte erzeugt werden. Es könnte z.B. sein, dass die Aktivität von Well-Known-System-Prozessen wie z.B. `conhost`- oder `svchost`-Prozessen heraus gefiltert wird. Werden diese Prozess jedoch Opfer einer Malware-Injection wäre ihre Aktivität sehr relevant. Daher kann eine zu starke Filterung für Probleme in der Erkennung sorgen [LB19, S. 36].

4.6 Verwundbarkeit

Sysmon nutzt für die Ereigniserkennung hauptsächlich Kernel-Callbacks, die beim Windows-Kernel selbst (Prozess-Manager, Objekt-Manager und Config-Manager) und bei Filtertreibern oder dem Filtermanager registriert sind. Abgesehen davon wird ETW für die Erkennung von Netzwerk-Ereignissen eingesetzt. Für den auf ETW basierenden Teil der Sensorik gelten die entsprechenden Angriffsmöglichkeiten bezüglich ETW, die im Kapitel 3 „Event Tracing for Windows“ im Abschnitt „Verwundbarkeit“ ausführlich beschrieben wurden. Für den größten Teil der Sensorik von Sysmon besteht die größte Gefahr jedoch in Angriffen auf das Kernel-Callback-System.

Solche Angriffe auf die Callback-Strukturen des Kernels sind aus Angriffen auf EDR-Agents bekannt. Die EDR-Endpunkt-Sensorik verwendet meist selbst Kernel-Callbacks in ihren Sicherheits-Treibern. Es gibt Anleitungen von Sicherheitsforschern und Penetration-Testern (Red-Team), in denen eine Manipulation der Kernel-Callback-Strukturen im Kernel beschrieben wird und in denen durch diese Methoden die EDR-Treiber gestört werden, sodass sie keine Ereignisse mehr erkennen können [Bau21][Ste20]. Diese Mechanismen sollten in gleicher Weise auch bei Sysmon funktionieren.

Manipulationen der Callback-Strukturen im Kernel sind nur aus dem Kernel-Mode möglich. Der Angreifer muss, um Sysmon auf diese Weise zu stören, in den Kernel eingedrungen sein bzw. die Möglichkeit haben, Schadcode im Kernel-Mode auszuführen. Aus dem User-Mode ist Sysmon-Sensorik im Kernel allgemein nicht angreifbar. Jedoch kann die User-Mode-Komponente der Sysmon-Software die Sysmon-Dienstanwendung auch aus dem User-Mode angegriffen werden [Bau21][Ste20].

Um eine vom Sysmon-Treiber (`SysmonDrv.sys`) registrierte Callback-Funktion zu entfernen, muss die Datenstruktur gefunden werden, indem die Kernel-Komponente oder der Filtermanager die registrierten Callback-Funktionen verwaltet. Solche Listen existieren in Form von Arrays mit Funktionspointern, welche die Adressen zu den registrierten Callback-Funktionen enthalten. Es existieren Methoden, diese Arrays für verschiedene Kernel-Callback-Mechanismen automatisiert im Kernel-Adressraum zu finden. Durch Prüfung der Funktions-Adressen in diesen Arrays auf ihr Ziel, bezogen auf die Information, in welchem Adressbereich dieses Ziel sich befindet, und einem Abgleich der Adressbereiche der geladener Kernel-Module, kann festgestellt werden, welche Funktionspointer auf Callback-Routinen des Sysmon-Treibers (`SysmonDrv.sys`) zeigen. Die Adressen dieser Funktionspointer können mit Null überschrieben werden. Dies ist möglich, da jeder Eintrag vor dem jeweiligen Aufruf von der Routine im Kernel geprüft wird, ob diese nicht mit Null belegt ist. Solchen Null-Pointern wird nicht gefolgt. Zudem werden die Callback-Arrays nicht durch PatchGuard geprüft. Auf diese Weise könnte der Sysmon-Treiber sehr stark in seiner Erkennung gestört werden, da seine registrierten Callback-Routinen nicht mehr aufgerufen werden. Eine noch elegantere Methode sieht vor, nicht den Funktionspointer im Array zu überschreiben, sondern den Code an dessen Ziel zu patchen. Dabei würde der Code der registrierten Callback-Routine im Code-Bereich des geladenen Sysmon-Treibers verändert. Dazu wird der Funktionspointer als Schreibadresse benutzt, an dessen Stelle geschrieben werden soll. Der Funktionspointer zeigt auf den ersten Maschinenbefehl der registrierten Callback-Routine. Durch ein Überschreiben dieser Speicherstelle mit dem Maschinenbefehl `RET` wird die Callback-Funktion im Sysmon-Treiber so verändert, dass sie beim Aufruf ohne das Ereignis zu registrieren sofort wieder verlassen wird. Sollten die Callback-Arrays durch PatchGuard oder eine andere Technik geschützt werden, wäre das eine alternative Möglichkeit einen solchen Schutzmechanismus zu umgehen. Die Daten und der Code von geladenen Kernel-Modulen sind generell nicht durch PatchGuard geschützt [Bau21][Ste20].

Angriffe über die Konfigurations- und Steuerungsschnittstelle (Kommandozeile) der Sysmon-Dienstanwendung im User-Mode, indem z.B. versucht wird, eine neue Konfiguration zu laden oder das Sysmon-Monitoring abzuschalten, versucht Sysmon zu protokollieren, sodass ein angeschlossenen Sicherheitssystem dieses mitbekommen und entsprechend reagieren könnte.

4.7 Nextron Aurora

Um Angriffe und Schadsoftware auf Basis protokollierter Systemereignisse zu erkennen, ist es notwendig in den Ereignisfolgen bestimmte Ereignisse oder Ereignismuster zu erkennen, die auf ein für einen Angriff auffälliges Verhalten hindeuten. Dies kann nachträglich auf Protokollen von Event-Objekten geschehen oder live auf den kontinuierlich eingehenden Event-Objekten eines laufenden Systems, das mittels einem Event-basierten Monitoring überwacht wird. Letzteres ist der in einer EDR-Lösung umgesetzte Anwendungsfall. Die Erkennung von relevanten Event-Objekten oder Gruppen-Mustern von Event-Objekten geschieht mittels Signaturen. Eine Signatur ist ein Satz von Filter-Regeln, mit denen die relevanten Event-Objekte in einem Log oder in einem Stream erkannt werden können. Eine Signatur kann dabei das Verhalten einer spezifische Malware-Familie oder ein allgemeineres schadhaftes Verhalten erkennen. Um möglichst weitreichend Angriffe und Malware verschiedenster Art zu erkennen, was der Anspruch von produktiver Endpunkt-Sensorik ist, ist es notwendig, die eingehenden Event-Objekte mit einer große Anzahl verschiedener Signaturen parallel zu filtern. So wie sich für die Erkennung statischer Artefakte das YARA-Format als Signaturformat etabliert hat, gibt es für die Signatur-basierte Analyse von dynamischen Event-Daten das SIGMA-Format als Standardformat für Event-Signaturen. Im Gegensatz zu statischen Signaturen sind dynamische Event-Signaturen in der Regel nicht unabhängig vom Event-System und den Event-Objekt-Formaten, da sich die Regeln in den Signaturen auf Ereignistypen und Attribut-Felder der Event-Objekte beziehen, die sich jedoch zwischen verschiedenen Event-Systemen (Sensorik-Komponenten) unterscheiden. Daher sind SIGMA-Signaturen für konkrete Ereignistypen bestimmter Event-Systeme zugeschnitten. Ein Großteil der SIGMA-Signaturen, die sich nicht auf spezielle kommerzielle Lösungen beziehen, sind für die Filterung von Event-Daten des Windows-Event-Logs oder Sysmon vorgesehen. Alleine für die Filterung von Sysmon-Events stehen über 1.000 Open-Source SIGMA-Signaturen kostenlos zur Verfügung [Rot21, min. 0:30]. Dadurch kommt Sysmon als Endpunkt-Sensorik eine größere Bedeutung zu, da für Eventdaten aus dieser Quelle große Mengen an Signaturen bestehen.

Die Sicherheitsfirma Nextron Systems des deutschen IT-Sicherheitsforschers und Unternehmers Florian Roth bietet eine Reihe von Softwarelösungen im Bereich der statischen und dynamischen Erkennung von Schadsoftware an. Dabei verfügt Nextron über große Bibliotheken von YARA- und SIGMA-Signaturen und große Kompetenz im Umgang mit signaturbasierten Scans. Eines der neusten Produkte von Nextron ist die Monitoring-Software Aurora für die ereignisbasierte Überwachung von Windows-Systemen [Rot21, min. 0:30][Rot22b, min. 0:55].

Aurora versucht als Endpunkt-Sensor (Endpoint-Agent) in etwa die gleichen Event-Objekte wie Sysmon zu liefern, sodass SIGMA-Signaturen für Sysmon-Events auch auf

die Events von Aurora angewendet werden können. Das besondere ist, dass Aurora als User-Mode-Anwendung ausgeführt wird und keine eigene Kernel-Komponenten benutzt. Stattdessen verwendet die Aurora-Software ausschließlich den in jedem Windows-System vorhandenen Kernel-Dienst Event-Tracing-for-Windows (ETW). Aurora erstellt dafür ETW-Sessions und bezieht entsprechende Eventdaten. Aus den Daten verschiedener ETW-Events und intern mitverfolgter Zustände versucht Aurora möglichst alle notwendigen Informationen zu ermitteln, um eigene Events zu generieren, die möglichst alle Informationen enthalten, die auch die Events der Sysmon-Software aufweisen. Zusätzlich ist Aurora in der Lage, noch bevor die Events in den Windows-Event-Log geschrieben werden oder zu einem externen SIEM oder EDR-System übermittelt werden, diese auf dem Endpunkt mittels SIGMA-Signaturen zu filtern [Rot21, min. 6:47][Rot22b, min. 4:00][Rot21, min. 3:30][Rot22a, S. 3][Rot22a, S. 4].

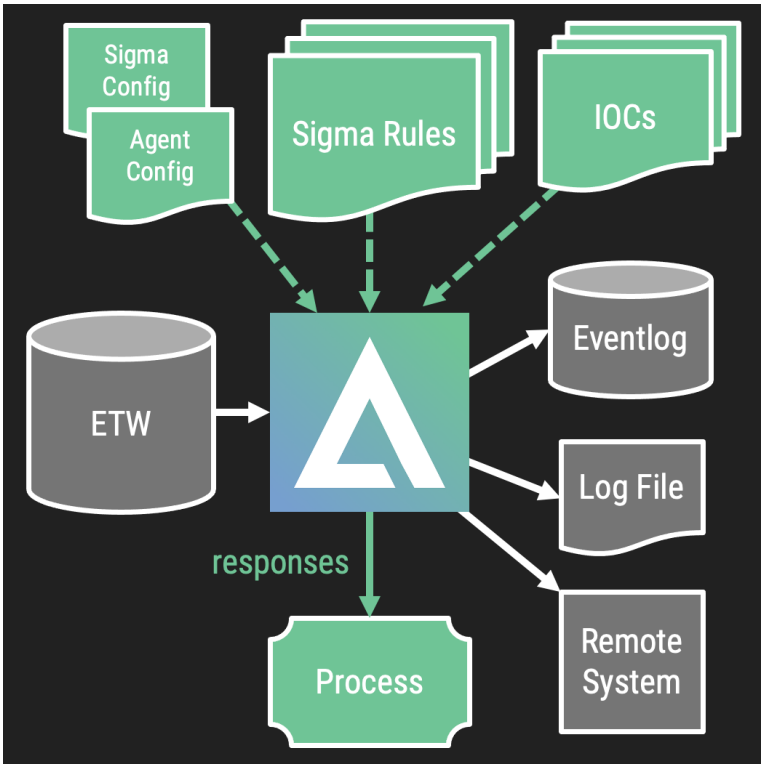

Abbildung 4.3: Nextron Aurora - Architektur [Rot22a, S. 3]

Aurora liefert durch die eigenen Ereignisse nach eigenen Angaben 70% der Informationen, die Sysmon bietet. Jedoch sind nicht alle Informationen der Sysmon-Ereignisse bezüglich der für Sysmon vorhandenen SIGMA-Signaturen gleich relevant. So schafft es Nextron nach eigenen Angaben mit Aurora eine Abdeckung von 95% der relevanten Ereignis-Informationen bezogen auf vorhandenen SIGMA-Signaturen im Vergleich zu Sysmon zu erreichen [Rot22a, S. 30]. Beispielsweise kann Aurora nach eigenen Angaben keine Named-Pipe-Aktivitäten erkennen, da diese Vorgänge keine ETW-Events erzeugen. Auch die Kommandozeilen-Argumente bei Prozesserzeugungen können in machen Fällen nicht

über ETW ermittelt werden, da die Informationen in den ETW-Events fehlen. Jedoch existieren auch nur sehr wenige SIGMA-Signaturen, welche diese Ereignis-Attribute in ihren Regeln nutzen. Nextron befindet sich trotzdem im Austausch mit Microsoft, um den Informationsgewinn mittels ETW weiter zu erhöhen [Rot22b, min. 11:52]. Aurora kann jedoch über ETW auch Vorgänge erkennen, die Sysmon nicht liefert. Aurora liefert jedoch auch Informationen, welche die Sysmon-Sensorik nicht bietet, daher ist der Vergleich bezüglich der Informationsmenge zwischen Sysmon und Aurora schwierig. Aurora liefert z.B. in allen Events als ein zusätzliches Attribut eine Liste alle Prozess-Namen des Prozessbaums, von dem der Prozess, der das Ereignis ausgelöst hat, abstammt. Ein solches Attribut, welches laut Nextron für die Filterung sehr relevant sei, bietet Sysmon nicht [Rot22a, S. 4][Rot22a, S. 30][Rot22b, min. 4:00][Rot21, min. 6:47].

Nextron sieht eine großen Stärke von Aurora darin, dass die Event-Objekte bereits auf den Endpunkten über SIGMA-Signaturen durchsucht werden können und nur die gefundenen Übereinstimmungen an ein zentrales SIEM weitergeleitet werden, anstatt alle Event-Objekte zu protokollieren oder weiterzuleiten, wie es bei Sysmon der Fall ist. Dies reduziert den Netzwerkverkehr und die im SIEM anfallende Datenmenge erheblich, was wiederum den Workload im SIEM reduziert [Rot22a, S. 5f][Rot21, min. 3:30][Rot22b, min. 11:52].

Aurora unterstützt SIGMA-Response-Actions, eine Erweiterung des SIGMA-Formats, welches neben Filter-Regeln zusätzlich erlaubt, Aktionen zu definieren, die ausgelöst werden, wenn die Regeln der Signatur anschlagen. So können z.B. Prozesse beendet (kill process) oder Speicherabzüge (Memory Dump) erstellt werden. Die Latenz bezüglich dem eingehenden ETW-Event und der Ausführung einer, in einer SIGMA-Regel konfigurierten, Reaktion, ist abhängig von der Menge der zu prüfenden SIGMA-Rules. Nextron zeigt Fälle, in dem eine Malware 200 ms bis 300 ms nachdem die ersten Events eingegangen sind, automatisch beendet werden konnte [Rot22a, S. 7][Rot22a, S. 33f][Rot22b, min. 17:09][Rot21, min. 10:36].

Neben der SIGMA-Filterung ermöglicht Aurora auch die Durchsuchung der Event-Daten mittels IOC-Signaturen. IOC (Indicators of Compromise) ist ein weiteres Signaturformat neben YARA und SIGMA. Während YARA-Regeln meist auf die Erkennung von Binaries spezialisiert sind, beziehen sind die Regeln in IOC-Signaturen meist auf Dinge der IT-Infrastruktur wie Dateinamen, DNS-Namen, Netzwerkadressen und den Inhalt von Netzwerkkommunikation [Rot22b, min. 33:30][Rot22a, S. 20].

Darüber hinaus kontrolliert Aurora die selbst verursachte CPU-Auslastung und beschränkt den Ressourcenverbrauch des eigenen Prozesses je nach konfiguriertem Limit. Dies kann jedoch bei großen Mengen an zu prüfenden Signaturen und einem zu niedrigen Limit dazu führen, dass nicht alle eingehenden ETW-Ereignisse verarbeitet werden können [Rot22b, min. 4:00][Rot22b, min. 11:52].

Sysmon vs. Aurora

Eigenschaft	Sysmon	Aurora
Datenquelle	- ETW - WMI-Filter - Sysmon-Driver — Kernel-Callbacks — Filter-Driver	only ETW
Kernel-Treiber notwendig	ja	nein
Risiken von Kernel-Problemen und Bluescreens (z.B. wenn mehrere Kernel-Sensoren parallel installiert sind)	ja	nein
Risiken von hoher Systemauslastung aufgrund des Overheads des Kernel-Drivers	ja	nein
Abdeckung Ereignistypen und Attribut-Felder	100%	70%
Abdeckung Ereignistypen und Attribut-Felder, die in SIGMA-Rules verwendet werden	100%	95%
relative Log-Größe	hoch	niedrig
integrierte SIGMA-Filterung	nein	ja
integrierte IOC-Filterung	nein	ja
SIGMA-Response-Actions	nein	ja
Resource-Limiter	nein	ja
Output: Windows EventLog	ja	ja
Output: Log-Datei	nein	ja (plain-Text/JSON)
Output: externer Dienst im Netzwerk (TCP/UDP-Target)	nein	ja (plain-Text/JSON)

Tabelle 4.29: Sysmon vs. Aurora [Rot22a, S. 4]

Es existieren zwei Versionen der Aurora-Software, eine kommerzielle und eine frei verfügbare Version. Letztere trägt den Namen Aurora-Agent-Lite und beinhaltet nicht alle Funktionen der kostenpflichtigen Vollversion. Der Lite-Version fehlt die Anbindung an Nextrons ASGARD-Management-System, über das SIGMA-Rules und Aurora-Konfigurationen in größeren Organisationen verwaltet und auf die Endpunkte aufgespielt werden können. Die kommerzielle Version enthält zudem spezielle nicht auf SIGMA basierende Erkennungsmodule, mit denen z.B. Cobalt-Strike-Angriffe, LSASS-Credential-Dumping und andere bösartige Vorgänge erkannt werden können. Jedem Nutzer der kostenpflichtigen Aurora-Version steht zudem der nicht öffentliche SIGMA-Feed von Nextron zur Verfügung, über den SIGMA-Signaturen für aktuelle Bedrohungen bereitgestellt werden [Rot21, min. 14:44][Rot22a, S. 31f].

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[Aar21] A. Aarness. Endpunktbasierte Detektion Und Reaktion (EDR), January 2021. CrowdStrike. https://www.crowdstrike.de/cybersecurity-101/endpoint-security/endpoint-detection-and-response-edr/. 1

[AIRS21] A. Allievi, A. Ionescu, M.E. Russinovich, and D.A. Solomon. *Windows Internals, Seventh Edition, Part 2*. Developer Reference. Pearson Education, 2021. ISBN: 9780135462409. 1.1.1, 1.1.2, 1.2.2, 1.2.3, 1.1, 1.4.2, 1.4.2.1, 1.5.4, 1.5.5, 1.5.5, 1.5.5, 1.5.6, 1.7.4, 2.4, 3.3, 3.5.2.2, 3.6, 3.6.1, 3.6.2, 3.6.3, 3.6.4, 3.5, 3.6.4, 3.6.4.1, 3.6.4.2, 3.6.4.3, 3.6.4.4, 3.6.5, 3.6.5, 3.7, 3.6.5, 3.6.6, 3.6.6, 3.6.7, 3.6.8, 3.6.9, 3.6.10, 3.6.11, 3.6.11, 3.6.12, 3.9.2.2, 3.9.2.7, 4.7

[Bau21] J. Bauters. Kernel Karnage, Part 1, October 2021. NVISO Labs. https://blog.nviso.eu/2021/10/21/kernel-karnage-part-1/. 1.5.11, 1.7.1, 1.7.2, 1.20, 1.7.2, 4.6, 4.7

[BfSidI] BSI Bundesamt für Sicherheit in der Informationstechnik. SiSyPHuS Win10: Studie zu Systemaufbau, Protokollierung, Härtung und Sicherheitsfunktion von Windows 10. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Cyber-Sicherheit/SiSyPHus/Workpackage2_Analyse_Gesamtsystem.html. 3.1

[Bro17] Z. Brown. Hidden Treasure: Detecting Intrusions with ETW, 2017. Derbycon 2017. http://www.irongeek.com/i.php?page=videos/derbycon7/t208-hidden-treasure-detecting-intrusions-with-etw-zac-brown. 3.1

[Cen] Defense Technical Information Center. AD1090036: MARPLE. https://apps.dtic.mil/sti/citations/AD1090036. 3.1

[Cha20a] G. Chappell. EVENT_TRACE_PROPERTIES, May 2020. https://www.geoffchappell.com/studies/windows/win32/advapi32/api/etw/logapi/event_trace_properties.htm . **??**, **??**, **??**, **??**, 3.14, 3.17, **??**, 3.27, **??**, 3.43, 3.44, 4.7

[Cha20b] G. Chappell. NtTraceControl, May 2020. https://www.geoffchappell.com/studies/windows/km/ntoskrnl/api/etw/traceapi/control/index.htm . 3.6.11

[Cha20c] G. Chappell. Trace Headers, January 2020. https://www.geoffchappell.com/studies/windows/km/ntoskrnl/api/etw/tracelog/traceheaders.htm . 3.9, 3.10, 3.6.8, 3.6.11, 4.7

[Che] Y. Chen. APTShield: A Real-Time Situation-Aware Detection System for Remote Access Trojan in the APT Attacks. DARPA. https://slideplayer.com/slide/12388286/. 3.1

[CMP21] R. Cobb, R. Mouse, and D. Panagiotopoulos. SharpSploit, August 2021. https://github.com/cobbr/SharpSploit. 3.9.1

[DAR] DARPA. Windows Low-Level System Monitoring Data Collection. https://slideplayer.com/slide/14288926. 3.1

[Des17] J. Desimone. Hunting for Memory-Resident Malware, 2017. Derbycon 2017. http://www.irongeek.com/i.php?page=videos/derbycon7/s21-hunting-for-memory-resident-malware-joe-desimone. 3.1

[EMY19] Everdox, N. Mulasmajic, and B. Yildiz. InfinityHook, July 2019. https://github.com/everdox/InfinityHook . 3.5.2, 3.5.2.1, 3.8.1, 3.9.2.7

[Gav18] H. Gavriel. Malware Mitigation when Direct System Calls are Used, November 2018. cyberbit. https://www.cyberbit.com/blog/endpoint-security/malware-mitigation-when-direct-system-calls-are-used/. 1.7.1

[Gol17] D. Goldshtein. Video: ETW - Monitor Anything, Anytime, Anywhere, July 2017. NDC Oslo 2017. https://www.youtube.com/watch?v=ZNdpLM4uIpw. 1.7.4, 3, 3.6, 3.6.4

[Han19] Clever Geek Handbook. Systemtimer in Windows, 2019. https://tech-de.netlify.app/articles/de522212/index.html. 1.1.2, 1.2.3, 3.5.2.2

[Har21a] O. Hartong. Sysmon 460 Schema, March 2021. https://gist.github.com/olafhartong/42b93050b6e0f49742cc3ea151d97f12. 4, 4.2.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 4.10, 4.11, 4.12, 4.13, 4.14, 4.15, 4.16, 4.17, 4.18, 4.19, 4.20, 4.21, 4.22, 4.23, 4.24, 4.25, 4.26, 4.27, 4.28, 4.7

[Har21b] O. Hartong. Sysmon Event Graph, November 2021. https://github.com/olafhartong/sysmon-cheatsheet/blob/master/Sysmon-graph.png. 4, 4.2, 4.7

[HH17] M. Hastings and D. Hull. Tracing Adversaries: Detecting Attacks with ETW, 2017. Derbycon 2017. http://www.irongeek.com/i.php?page=videos/derbycon7/s25-tracing-adversaries-detecting-attacks-with-etw-matt-hastings-dave-hull. 3.1

[HL21] H. Hiroaki and T. Lee. Earth Baku Returns, August 2021. Trend Micro Inc. https://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/earth-baku-returns. 3.9.1

[Hos17] A. Hosseini. Ten process injection techniques: A technical survey of common and trending process injection techniques, July 2017. Elastic Tech Topics. https://www.elastic.co/de/blog/ten-process-injection-techniques-technical-survey-common-and-trending-process. 1.5.2, 1.5.10

[Lab18] Kaspersky Lab. The Slingshot APT, March 2018. https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2018/03/09133534/The-Slingshot-APT_report_ENG_final.pdf. 3.9.1

[LB19] S. Le Berre. Sysmon Internals, July 2019. Exa Track. https://exatrack.com/public/SysmonInternals.pdf. 1.7.3, 4, 4.1.1, 4.3, 4.3.1, 4.3.2, 4.3.3, 4.3.4, 4.3.5, 4.3.6, 4.3.7, 4.3.8, 4.5

[MA] MITRE-ATT&CK. T1562.006: Impair Defenses - Indicator Blocking. https://attack.mitre.org/techniques/T1562/006/. 3.9.1

[MicCg] Microsoft Corp. *Windows API Documentation Consoles*, 2020 Cg. https://docs.microsoft.com/en-us/windows/console/consoles. 1.5.7

[Mic2a] Microsoft Corp. *Windows Server Commands Documentation logman*, 2021 2a. https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/logman. 3.6.2, 3.6.10.3

[Mic2e] Microsoft Corp. *Windows API Documentation ETW - Message Compiler*, 2021 2e. https://docs.microsoft.com/en-us/windows/win32/wes/message-compiler--mc-exe-. 3.6.9.1

[Mic4b] Microsoft Corp. *Windows API Documentation Time - System Time*, 2021 4b. https://docs.microsoft.com/en-us/windows/win32/sysinfo/system-time. 3.5.2.2

[Mic4c] Microsoft Corp. *Windows API Documentation Time - Interrupt Time*, 2021 4c. https://docs.microsoft.com/en-us/windows/win32/sysinfo/interrupt-time. 1.2.3, 3.5.2.2

[Mic4d] Microsoft Corp. *Windows API Documentation Time - File Time*, 2021 4d. https://docs.microsoft.com/en-us/windows/win32/sysinfo/file-times. 1.4.2.1

[Mic5a] Microsoft Corp. *Windows API Documentation ETW - Event Tracing Structures*, 2021 5a. https://docs.microsoft.com/en-us/windows/win32/api/_etw/#structures. 3.5.1, 3.5.2, 3.5.2.1, 3.5.2.2, 3.5.2.2, 3.5.2.3, 3.6.5, 3.6.11

[Mic5b] Microsoft Corp. *Windows API Documentation ETW - WNODE_HEADER*, 2021 5b. https://docs.microsoft.com/en-us/windows/win32/etw/wnode-header. 1.2.3, 3.5.2, 3.5.2.1, 3.5.2.2, 3.5.2.2, 3.5.2.3

[Mic5c] Microsoft Corp. *Windows API Documentation ETW - EVENT_RECORD*, 2021 5c. https://docs.microsoft.com/en-us/windows/win32/api/evntcons/ns-evntcons-event_record. 3.6.7, 3.9, 3.10, 3.6.11, 4.7

[Mic5d] Microsoft Corp. *Windows API Documentation ETW - EVENT_HEADER*, 2021 5d. https://docs.microsoft.com/en-us/windows/win32/api/evntcons/ns-evntcons-event_header. 3.9, 3.10, 3.2, 4.7

[Mic5e] Microsoft Corp. *Windows API Documentation ETW - EVENT_DESCRIPTOR*, 2021 5e. https://docs.microsoft.com/en-us/windows/win32/api/evntprov/ns-evntprov-event_descriptor. 1.4.3.1, 3.9, 3.10, 4.7

[Mic5f] Microsoft Corp. *Windows API Documentation ETW - EVENT_TRACE_PROPERTIES*, 2021 5f. https://docs.microsoft.com/en-us/windows/win32/api/evntrace/ns-evntrace-event_trace_properties. 3.12, 3.13, 3.14, 3.15, 3.16,

3.17, 3.18, 3.19, 3.20, 3.21, 3.22, 3.23, 3.24, 3.25, 3.26, 3.27, 3.28, 3.29, 3.30, 3.31, 3.32, 3.33, 3.34, 3.35, 3.36, 3.37, 3.38, 3.39, 3.40, 3.41, 3.42, 3.43, 3.44, 3.45, 4.7

[Mic6a] Microsoft Corp. *Windows API Documentation ETW - About Event Tracing*, 2021 6a. https://docs.microsoft.com/en-us/windows/win32/etw/about-event-tracing. 3, 3.3, 3.6, 3.6.2, 3.6.3, 3.6.4

[Mic6b] Microsoft Corp. *Windows API Documentation ETW - Event Tracing Sessions*, 2021 6b. https://docs.microsoft.com/en-us/windows/win32/etw/event-tracing-sessions. 3.5.2, 3.6.4, 3.6.4.1, 3.6.4.2, 3.6.5

[Mic7a] Microsoft Corp. *Windows API Documentation ETW - Writing an Instrumentation Manifest*, 2021 7a. https://docs.microsoft.com/en-us/windows/win32/wes/writing-an-instrumentation-manifest. 3.6.4, 3.1

[Mic8a] Microsoft Corp. *Windows API Documentation ETW - Providing Events*, 2021 8a. https://docs.microsoft.com/en-us/windows/win32/etw/providing-events. 3.6.1, 3.6.9

[Mic8b] Microsoft Corp. *Windows API Documentation ETW - Writing Manifest-based Events*, 2021 8b. https://docs.microsoft.com/en-us/windows/win32/etw/writing-manifest-based-events. 3.6.1, 3.6.4.1, 3.6.7, 3.6.9.1

[Mic8c] Microsoft Corp. *Windows API Documentation ETW - Writing MOF (Classic) Events*, 2021 8c. https://docs.microsoft.com/en-us/windows/win32/etw/tracing-events. 3.6.4.2, 3.6.9.2

[Mic8d] Microsoft Corp. *Windows Driver Documentation ETW - Adding Event Tracing to Kernel-Mode Drivers*, 2021 8d. https://docs.microsoft.com/en-us/windows-hardware/drivers/devtest/adding-event-tracing-to-kernel-mode-drivers. 3.6.9.1, 3.6.9.3

[Mic8e] Microsoft Corp. *Windows Driver Documentation ETW - WMI Event Tracing*, 2021 8e. https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/wmi-event-tracing. 3.6.9.4

[Mic9a] Microsoft Corp. *Windows API Documentation ETW - Configuring and Starting an Event Tracing Session*, 2021 9a. https://docs.microsoft.com/en-us/windows/win32/etw/configuring-and-starting-an-event-tracing-session. 3.6.2, 3.6.5, 3.6, 3.6.8, 3.6.10.1, 3.11, 4.7

[Mic9b] Microsoft Corp. *Windows API Documentation ETW - Configuring and Starting a SystemTraceProvider Session*, 2021 9b. https://docs.microsoft.com/en-us/windows/win32/etw/configuring-and-starting-a-systemtraceprovider-session. 3.6.6, 3.6.10.2, 3.6.10.3

[Mic9c] Microsoft Corp. *Windows API Documentation ETW - Configuring and Starting the NT Kernel Logger Session*, 2021 9c. https://docs.microsoft.com/en-us/windows/win32/etw/configuring-and-starting-the-nt-kernel-logger-session . 3.6.5, 3.6.6, 3.6.6, 3.6.10.2

[Mic9d] Microsoft Corp. *Windows API Documentation ETW - Configuring and Starting an AutoLogger Session*, 2021 9d. https://docs.microsoft.com/en-us/windows/win32/etw/configuring-and-starting-an-autologger-session . 3.6.12, 3.3, 3.6.12, 3.9.2.2

[MicAa] Microsoft Corp. *Windows API Documentation ETW - Consuming Events (Event Tracing)*, 2021 Aa. https://docs.microsoft.com/en-us/windows/win32/etw/consuming-events. 3.6, 3.6.3, 3.6.8, 3.6.11

[MicAb] Microsoft Corp. *Windows API Documentation ETW - Retrieving Event Data*, 2021 Ab. https://docs.microsoft.com/en-us/windows/win32/etw/retrieving-event-data. 3.6.8, 3.6.11

[MicBa] Microsoft Corp. *Windows API Documentation ETW - System Providers*, 2021 Ba. https://docs.microsoft.com/en-us/windows/win32/etw/system-providers. 3.3, 3.6.6, 3.6.6

[MicBb] Microsoft Corp. *Windows API Documentation ETW - EVENT_TRACE_PROPERTIES_V2 structure*, 2021 Bb.

https://docs.microsoft.com/en-us/windows/win32/api/evntrace/ns-evntrace-event_trace_properties_v2. 3.6.6, 3.6.6

[MicCa] Microsoft Corp. *Windows API Documentation System - Processes and Threads*, 2021 Ca. https://docs.microsoft.com/en-us/windows/win32/procthread/processes-and-threads. 1.4.1.1, 1.5.1, 1.5.2

[MicCb] Microsoft Corp. *Windows API Documentation System - Child Processes*, 2021 Cb. https://docs.microsoft.com/en-us/windows/win32/procthread/child-processes. 1.5.1

[MicCe] Microsoft Corp. *Windows API Documentation System - Interprocess Communications*, 2021 Ce. https://docs.microsoft.com/en-us/windows/win32/ipc/interprocess-communications. 1.5.6

[Mic1a] Microsoft Corp. *Sysinternals Documentation Sysmon v13.33*, 2022 1a. https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon. 4, 4.1, 4.1.1, 4.1.2, 4.1.3, 4.1.4, 4.1, 4.1.5, 4.1.6, 4.1.7, 4.2.2, 4.7

[Mic2b] Microsoft Corp. *Windows Commands Documentation xperf*, 2022 2b. https://docs.microsoft.com/en-us/windows-hardware/test/wpt/xperf-command-line-reference. 3.6.2, 3.6.10.3

[Mic2c] Microsoft Corp. *Windows PowerShell Documentation Event Tracing Management*, 2022 2c. https://docs.microsoft.com/en-us/powershell/module/eventtracingmanagement/?view=windowsserver2022-ps . 3.6.2, 3.6.10.3

[Mic2d] Microsoft Corp. *Windows Server Commands Documentation wevtutil*, 2022 2d. https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/wevtutil. 3.6.4, 3.6.10.3

[Mic3a] Microsoft Corp. *Windows API Documentation PCW - About Performance Counters*, 2022 3a. https://docs.microsoft.com/en-us/windows/win32/perfctrs/about-performance-counters. 2.1

[Mic4a] Microsoft Corp. *Windows API Documentation Time - Acquiring high-resolution time stamps*, 2022 4a. https://docs.microsoft.com/en-us/windows/win32/sysinfo/acquiring-high-resolution-time-stamps. 1.2.3, 3.5.2.1

[Mon22] S. Monk. Understanding Sysmon Events using SysmonSimulator, January 2022. RootDSE. https://rootdse.org/posts/understanding-sysmon-events/. 4, 4.2.1, 4.2.2, 4.3, 4.3.1, 4.3.2, 4.3.3, 4.3.4, 4.3.6, 4.3.7, 4.3.8, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 4.10, 4.11, 4.12, 4.13, 4.14, 4.15, 4.16, 4.17, 4.18, 4.19, 4.20, 4.21, 4.22, 4.23, 4.24, 4.25, 4.26, 4.27, 4.28, 4.7

[Mos21] F. Mosch. A tale of EDR bypass methods, January 2021. https://s3cur3th1ssh1t.github.io/A-tale-of-EDR-bypass-methods/. 1.7.1, 1.7.2

[OS] Optiv-Security. ScareCrow. https://github.com/optiv/ScareCrow. 3.9.1

[OSD19] OSDev. Time And Date, 2019. https://wiki.osdev.org/Time. 1.1.2

[PHAK21] C. Peres, O. Hartong, and T. Al Khatib. TrustedSec Sysmon Community Guide, December 2021. https://github.com/trustedsec/SysmonCommunityGuide. 4, 4.1.1, 4.1.2, 4.1.3, 4.1.4, 4.1.5, 4.1.6, 4.1.7, 4.2.1, 4.2.2

[PL20] S. Peyrefitte and A. Larson. etl-parser, July 2020. Airbus CERT. https://github.com/airbus-cert/etl-parser. 3.6.8

[Reu20] M. Reuter. Möglichkeiten zum Einsatz von Event Tracing for Windows (ETW) zur Unterstützung forensischer Analysen von Prozessverhalten in Windows 10, August 2020. Hochschule Wismar. https://it-forensik.fiw.hs-wismar.de/images/a/a3/MT_MReuter.pdf. 3

[Rot21] F. Roth. Video: Nextron Aurora - Free Sigma-based EDR Agent Preview, November 2021. Nextron Systems. https://www.youtube.com/watch?v=ONuK9NIXoOs. 1.7.4, 4.7, 4.7, 4.7

[Rot22a] F. Roth. Slides: Aurora Agent, April 2022. Nextron Systems. https://www.nextron-systems.com/wp-content/uploads/2022/04/Aurora_Agent_Overview_EN_2022_Mar.pdf. 4.7, 4.3, 4.7, 4.29, 4.7, 4.7, 4.7

[Rot22b] F. Roth. Video: Aurora Pre-Release Session, April 2022. Nextron Systems. https://www.youtube.com/watch?v=qUAhkyIax3k. 1.7.4, 4.7, 4.7

[Rus12] M.E. Russinovich. Malware Hunting with the Sysinternals Tools, 2012. Microsoft TechEd 2012. video.ch9.ms/teched/2012/eu/SIA302.pptx. 3.1

[SBK+] B.E. Storm, J.A. Battaglia, M.S. Kemmerer, W. Kupersanin, D.P. Miller, C. Wampler, S.M. Whitley, and R.D. Wolf. Finding Cyber Threats with ATT&CK-Based Analytics. MITRE. https://www.mitre.org/sites/default/files/publications/16-3713-finding-cyber-threats%20with%20att&ck-based-analytics.pdf. 3.1

[Sha20] Y. Shafir. Exploiting a "Simple" Vulnerability – In 35 Easy Steps or Less!, November 2020. Windows Internals https://windows-internals.com/exploiting-a-simple-vulnerability-in-35-easy-steps-or-less/. 3.9.2.8

[Sol20] S. Solnica. Fixing empty paths in FileIO events (ETW), August 2020. Low Level Design. https://lowleveldesign.org/2020/08/15/fixing-empty-paths-in-fileio-events-etw/. 1.6.1, 3.6.13

[Ste20] Z. Stein. Blinding EDR On Windows, October 2020. https://synzack.github.io/Blinding-EDR-On-Windows/. 1.7.1, 1.7.2, 1.20, 1.7.2, 4.6, 4.7

[TKG21a] C. Teodorescu, I. Korkin, and A. Golchikov. Slides: Veni, No Vidi, No Vici - Attacks on ETW Blind EDR Sensors, November 2021. BlackHat Europe 2021. https://i.blackhat.com/EU-21/Wednesday/EU-21-Teodorescu-Veni-No-Vidi-No-Vici-Attacks-On-ETW-Blind-EDRs.pdf. 3, 3.1, 3.6.4, 3.6.6, 3.6.9.3, 3.8, 3.8.1, 3.9, 3.9, 3.9.1, 3.9.2.1, 3.9.2.2, 3.9.2.3, 3.9.2.4, 3.9.2.5, 3.9.2.6, 3.9.2.7

[TKG21b] C. Teodorescu, I. Korkin, and A. Golchikov. Video: Veni, No Vidi, No Vici - Attacks on ETW Blind EDR Sensors, November 2021. BlackHat Europe 2021. https://youtube.com/watch?v=wZG0h1q7fMg. 1.7.4, 3, 3.1, 3.6.6, 3.8, 3.8.1, 3.9, 3.8, 3.9, 3.9.1, 3.9.2.1, 3.9.2.2, 3.9.2.3, 3.9.2.4, 3.9.2.5, 3.9.2.6, 3.9.2.7, 4.7

[Wik18] Wikipedia. Event Tracing for Windows, Januar 2018. https://de.wikipedia.org/wiki/Event_Tracing_for_Windows. 3.2

[Wik20a] Wikipedia. Interprozesskommunikation, November 2020. https://de.wikipedia.org/wiki/Interprozesskommunikation. 1.5.6

[Wik20b] Wikipedia. Security Identifier, December 2020. https://de.wikipedia.org/wiki/Security_Identifier. 1.2, 1.4.1.1, 4.7

[Wik21a] Wikipedia. Mandatory Integrity Control, January 2021. https://en.wikipedia.org/wiki/Mandatory_Integrity_Control. 1.4.1.1, 1.4.2.1, 1.5.10

[Wik21b] Wikipedia. Registrierungsdatenbank, May 2021. https://de.wikipedia.org/wiki/Registrierungsdatenbank. 1.5.5, 1.7, 1.8, 1.5.5, 1.9, 4.7

[Wik22a] Wikipedia. Desktop Window Manager, March 2022. https://en.wikipedia.org/wiki/Desktop_Window_Manager. 1.5.7

[Wik22b] Wikipedia. Inter-Process Communication, March 2022. https://en.wikipedia.org/wiki/Inter-process_communication. 1.5.6

[Wik22c] Wikipedia. Microsoft Windows library files, March 2022. https://en.wikipedia.org/wiki/Microsoft_Windows_library_files. 1.7.1

[Wik22d] Wikipedia. Security Identifier, February 2022. https://en.wikipedia.org/wiki/Security_Identifier. 1.4.1.1

[Wik22e] Wikipedia. Universally unique identifier, April 2022. https://en.wikipedia.org/wiki/Universally_unique_identifier. 1.4.1.1

[Wik22f] Wikipedia. Windows API, April 2022. https://en.wikipedia.org/wiki/Windows_API. 1.7.1

[Wik22g] Wikipedia. Windows Registry, May 2022. https://en.wikipedia.org/wiki/Windows_Registry. 1.5.5, 1.5.5, 1.5.5

[YIRS17] P. Yosifovich, A. Ionescu, M.E. Russinovich, and D.A. Solomon. *Windows Internals, Seventh Edition, Part 1*. Developer Reference. Pearson Education, 2017. ISBN: 9783864905384. 1.1.1, 1.1.2, 1.1.3, 1.2.3, 1.4.1.1, 1.4.1.1, 1.4.1.1, 1.4.2.1, 1.5.2, 1.5.3, 1.5.4, 1.5.5, 1.5.5, 1.5.11, 1.7.1, 1.7.2, 1.7.3, 2.1, 3.8, 3.8.1, 3.9.2.6

[Yos19] P. Yosifovich. Video: Building your own profiling and diagnosis tools with Event Tracing for Windows, August 2019.
DotNext. `https://www.youtube.com/watch?v=gBkvA002qUY`. 1.7.4, 2.1, 3, 3.2, 3.6

[Yos20] P. Yosifovich. *Windows Kernel Programming*.
Leanpub by Pavel Yosifovich, 2020. ISBN: 9781977593375. 1.1.1, 1.1.2, 1.2.2, 1.4.1.1, 1.5.4, 1.5.11, 1.7.2, 1.7.3, 4.3

[Yos21] P Yosifovich. ETW Explorer, August 2021.
`https://github.com/zodiacon/EtwExplorer`. 3.6.4