

Master-Thesis

Möglichkeiten zum Einsatz von Event Tracing for Windows (ETW)
zur Unterstützung forensischer Analysen von Prozessverhalten
in Windows 10

Eingereicht am: 31. August 2020

von: Martin Reuter

Erstgutachterin: Prof. Dr.-Ing. Antje Raab-Düsterhöft

Zweitgutachter: Dipl.-Ing. Hans-Peter Merkel

Betreuer: Dipl.-Inf. Maximilian Winkler
(Bundesamt für Sicherheit in der
Informationstechnik)

Titel

Möglichkeiten zum Einsatz von Event Tracing for Windows (ETW) zur Unterstützung forensischer Analysen von Prozessverhalten in Windows 10

Titel (engl.)

Possibilities of using Event Tracing for Windows (ETW) to assist the Forensic Analysis of Process Behavior in Windows 10

Aufgabenstellung

Im Rahmen der Master-Thesis soll untersucht werden, ob und wie sich Event Tracing for Windows (ETW) einsetzen lässt, das Verhalten von Prozessen in Windows 10 zu analysieren und sich die hieraus erzielten Erkenntnisse zur Unterstützung forensischer Analysen nutzen lassen.

Nach eingehender Betrachtung der – insbesondere aus forensischer Sicht – relevanten Verhaltensweisen von Prozessen in Windows 10 werden bisherige Forschungsergebnisse zum Einsatz von ETW im forensischen Umfeld vorgestellt. Eine Identifizierung und Betrachtung von (forensisch) relevanten ETW-Providern soll mit Hilfe bereits bestehender Forschungsergebnisse sowie auf Grundlage eines zu definierenden Test-szenarios vorgenommen werden. Die Informationen werden anschließend in einer strukturierten Vorgehensweise zur Prozessanalyse zusammengeführt und beschrieben. Für einen konkreten Anwendungsfall wird ein Demonstrator (Proof-of-Concept) entwickelt.

Mögliche Grenzen und Herausforderungen sollen dargelegt und aufgezeigt werden. Abschließend sollen die erzielten Ergebnisse sowie gewonnenen Erkenntnisse, in Hinblick auf ihre Eignung zur Unterstützung forensischer Analysen von Prozessverhalten, einer Bewertung unterzogen werden.

Kurzreferat

Die Master-Thesis gibt einen Überblick über die Möglichkeiten zum Einsatz von Event Tracing for Windows (ETW) zur Unterstützung forensischer Analysen von Prozessverhalten in Windows 10.

Es wird der Aufbau und die Funktionsweise von ETW mit ihren Schnittstellen betrachtet und die Programmierschnittstelle verwendet, um von ETW-Providern gelieferte Events in einer Analysesitzung aufzuzeichnen. Die aufgezeichneten Events werden einem zu analysierenden Prozess zugeordnet und zur Bestimmung von Prozessverhaltensweisen ausgewertet. Hierbei lassen sich neben den bereits in anderen Forschungsarbeiten und Projekten gesammelten Erkenntnisse heranziehen, um (weitere) forensisch relevante ETW-Provider zu identifizieren. Durch den Einsatz von ETW wird eine Verhaltensanalyse von Prozessen, wie beispielsweise einer ausgeführten Malware, ermöglicht. Die bezogenen Informationen werden dabei ausschließlich aus den bereitgestellten Systemschnittstellen des Betriebssystems Windows 10 gewonnen, sodass keine zusätzlichen Komponenten, wie Kernel-Mode Treiber, implementiert werden müssen.

Abstract

This Master-Thesis gives an overview of using Event Tracing for Windows (ETW) to assist the forensic analysis of process behaviour in Windows 10.

Architecture and functionality of ETW and its interfaces are being examined to capture events delivered by any ETW-Provider to an analysis ETW-Session. The captured events are used to determine the specific behaviour of an observed process. Previous research findings and already existing projects on ETW provide a solid basis to determine further ETW-Providers, which can assist forensic analysis. ETW offers the opportunity to analyse process behaviour, i.e. of malware processes, by capturing data that is provided from Windows 10 itself. By using ETW as a forensic data source for gathering information on process behavior, there is no requirement for implementing additionally components, i.e. Kernel-Mode drivers.

Inhaltsverzeichnis

1	Motivation	8
2	Einleitung	10
2.1	Ausgangssituation	10
2.2	Zielsetzung	11
2.3	Aufbau der Arbeit	11
3	Definitionen und Grundlagen	13
3.1	Prozesse und Threads	13
3.2	Eltern- und Kindprozesse	14
3.3	Prozesserzeugung unter Windows	14
3.4	Virtueller Speicher	15
3.5	User- und Kernel-Mode	15
3.6	(System-)Programmierschnittstellen	16
3.6.1	Windows API (Win32-API)	16
3.6.2	Windows Native API	16
3.7	Verhalten von (Malware-)Prozessen	16
3.7.1	Erstellen von (weiteren) Prozessen und Threads	17
3.7.2	Ein- und Ausbinden von Bibliotheken (DLLs)	17
3.7.3	Ein- und Ausgabeoperationen (E/A)	17
3.7.4	Dateisystemzugriffe	18
3.7.5	Windows-Registrierungsdatenbank (Registry)	18
3.7.6	Inter-Prozesskommunikation	19
3.7.7	Weitere Funktionsaufrufe	22
3.8	Windows Logging-Funktionalitäten	23
3.8.1	Windows Event Logging (Ereignisprotokollierung)	23
3.8.2	Event Tracing for Windows (ETW)	24
3.9	ETW als Framework in Windows	25
3.9.1	Übersicht	25
3.9.2	Architektur	26
3.9.3	Informationsfluss	30
3.9.4	Aufbau der Eventstruktur	31
3.9.5	Abgrenzung zum Windows EventLog	32
4	Stand der Technik / Forschung	33
4.1	Einsatzmöglichkeiten von Event Tracing for Windows (ETW)	33
4.1.1	Debugging	33
4.1.2	Performanz- und Leistungsanalyse	33
4.1.3	Windows Telemetrie	34

4.1.4	Threat Hunting	36
4.1.5	IT-Forensik	37
4.2	Schnittstellen zu ETW in Windows 10	38
4.2.1	Computer Management - Performance (GUI)	38
4.2.2	Logman (CLI)	38
4.2.3	Windows Events Command Line Utility (CLI)	39
4.2.4	Event Tracing API (Win32-API)	39
4.3	Werkzeuge und Hilfsmittel für ETW	40
4.3.1	Wrapper	40
4.3.2	Werkzeuge	41
4.3.3	Analysetools und Frameworks	44
4.4	Herausforderungen	51
4.4.1	Identifikation von ETW-Providern	51
4.4.2	Konsumierung von ETW-Events	52
4.4.3	Auswertung und Bewertung der Informationen	53
5	Vorgehensweise zur Analyse von Prozessverhalten	55
5.1	Vorbereitung und Fokus der Analyse	55
5.2	Methodik der Aufzeichnung	58
5.3	Identifikation und Auswahl geeigneter ETW-Provider	60
5.4	Aufzeichnen des Prozessverhaltens	62
5.4.1	Anlegen der ETW-Analysesitzung	62
5.4.2	Zuordnen von ETW-Providern zur Analysesitzung	62
5.4.3	Konfiguration der Aufzeichnungsmethodik	62
5.4.4	Starten der Aufzeichnung durch Aktivierung der Analysesitzung	63
5.4.5	Prozess starten und überwachen	63
5.4.6	Aufzeichnung beobachten und Events filtern	63
5.4.7	Stoppen der Aufzeichnung nach Terminierung des Prozesses	64
5.5	Auswertung der Aufzeichnung	64
5.6	Bewertung der Ergebnisse	65
6	Proof-of-Concept (PoC): Prozessanalyse mit Hilfe von ETW	66
6.1	Exemplarischer Anwendungsfall: DemoProcess (PoC I)	66
6.1.1	Vorbereitung und Fokus der Analyse	69
6.1.2	Methodik der Aufzeichnung	71
6.1.3	Identifikation und Auswahl geeigneter ETW-Provider	71
6.1.4	Aufzeichnen des Prozessverhaltens	77
6.1.5	Auswertung der Aufzeichnung	83
6.1.6	Bewertung der Ergebnisse	91
6.2	Konkreter Anwendungsfall: njRAT (PoC II)	91
6.2.1	Vorbereitung und Fokus der Analyse	92
6.2.2	Methodik der Aufzeichnung	92
6.2.3	Identifikation und Auswahl geeigneter ETW-Provider	93
6.2.4	Aufzeichnen des Prozessverhaltens	93
6.2.5	Auswertung der Aufzeichnung	95
6.2.6	Bewertung der Ergebnisse	101

6.3	Konkreter Anwendungsfall: Emotet (PoC III)	102
6.3.1	Vorbereitung und Fokus der Analyse	103
6.3.2	Methodik der Aufzeichnung	103
6.3.3	Identifikation und Auswahl geeigneter ETW-Provider	103
6.3.4	Aufzeichnen des Prozessverhaltens	104
6.3.5	Auswertung der Aufzeichnung	106
6.3.6	Bewertung der Ergebnisse	123
7	Grenzen und Herausforderungen	126
8	Bewertung	130
9	Zusammenfassung und Ausblick	133
	Literaturverzeichnis	135
	Abbildungsverzeichnis	145
	Listingverzeichnis	146
	Tabellenverzeichnis	147
	Abkürzungsverzeichnis	148
	 Anhang	
A	Auflistung von Quellcode	150
A.1	PoC-Analysewerkzeug: ETWProcessTracer (v.0.1)	150
A.2	Exemplarischer Anwendungsfall: DemoProcess (PoC I)	157
A.3	Hilfstool: ProcessStartHelper	161
B	Weitere Bilder	162
B.1	Exemplarischer Anwendungsfall: DemoProcess (PoC I)	162
B.2	Hilfstool - ProcessStartHelper	162
B.3	PoC-Analysewerkzeug - ETWProcessTracer (v.0.1)	163
B.4	Analyseframework: SilkETW (v.0.8)	164
B.5	PowerKrabsETW (v.0.1)	164
B.6	Analysetool: PSWasp (v.0.0.0.1)	165
B.7	ProcMonX (0.21 Beta)	165
C	Windows Kernel Trace (NT Kernel Logger)	166
C.1	Zuordnung der Kernel Keywords (ETW-API) zur TraceEvent Library	166
C.2	Kernel Keyword-Collections (TraceEvent Library)	167
C.3	Kernel Keyword-Collection: Forensic (ETWProcessTracer)	167
D	Forensisch relevante ETW-Provider aus Kap. 6 (PoC)	168
D.1	Prozesse und Threads	168
D.2	Dateisystem	169
D.3	Windows-Registry	170

D.4	Netzwerk und Kommunikation	171
D.5	Skripting	174
E	Konfigurationsparameter der Analyseumgebung	175
E.1	Virtualisierungsumgebung	175
E.2	Analysesystem (VM): Windows 10 Version 1909 (Education)	176
	Selbstständigkeitserklärung	178
	Thesen	179

1 Motivation

Betriebssysteme stellen eine zentrale Schnittstelle zwischen Geräten (Hardware) und Anwendungsprogrammen (Software) von Benutzenden her. Innerhalb moderner Betriebssysteme stellen unterschiedliche Konzepte eine effiziente Verwaltung von Aktivitäten sicher, um die zur Verfügung stehenden Ressourcen (Prozessor, Arbeitsspeicher, Ein- und Ausgabegeräte) optimal einsetzen und auslasten zu können [TB15, S. 1]. Dabei werden die Funktionalitäten gängiger Betriebssysteme häufig an bestimmten Einsatzzwecke (z.B. als Desktop-PC, Server, Smartphone oder Tablet) ausgerichtet. Grundsätzlich wird ein Betriebssystem immer benötigt, um eine Interaktion zwischen Hardware und Software herzustellen und zu ermöglichen.

Der mögliche Funktionsumfang ausgeführter Software wird durch das Betriebssystem limitiert und über zentral bereitgestellte (System-)Programmierschnittstellen (engl. *Application Programming Interface*, API) bereitgestellt. Die zur Verfügung stehenden Funktionen einer API sollten für Softwareentwickelnde in ausreichendem Umfang dokumentiert sein. Betriebssysteme stellen als zentralen Mittelpunkt und Schnittstelle zwischen Hard- und Software einen wichtigen Ausgangspunkt für forensische Analysen dar [Bun11, S. 65-67].

Im Bereich der Arbeitsplatzcomputer (Desktop-PC, Laptop) liegen Microsoft Windows Betriebssysteme im Januar 2020 weltweit mit einem Marktanteil von 77,7 % vor Apple macOS X (17,07 %) und Linux (1,9 %) [Ten20]. Durch den hohen Marktanteil stehen Windows Betriebssysteme häufig im Fokus von Angreifenden und stellen ein beliebtes Ziel für Angriffe dar. Windows Betriebssysteme werden durch Microsoft als proprietäre Software bereitgestellt – der Quellcode steht Dritten nicht zur Verfügung, sodass sich die Möglichkeiten einer Interaktion und Implementierung von Software aus den betriebssystemseitig bereitgestellten Schnittstellen beschränken. Die verfügbaren Schnittstellen und Funktionen sind häufig nicht oder nur unzureichend dokumentiert. Neben Softwareentwickelnden erschwert dies auch die Arbeit für IT-Forensiker/-innen, Analytinnen und Analytisten oder Incident Respondern und stellt sie vor große Herausforderungen.

Aus Sicht der IT-Forensik stehen Untersuchungen über Verhalten und Aktivitäten von berechtigten Benutzenden oder unberechtigten Dritten, wie beispielsweise Angreifenden, in einem besonderen Fokus. Es müssen daher definierte Schnittstellen existieren und den Analysierenden bekannt gemacht werden, aus denen relevante Informationen über Aktivitäten innerhalb des Betriebssystems und ausgeführter Software bezogen werden können.

Im Bereich der Speicherforensik ist die Auswertung von Prozess- und Speicherabbildern zur Analyse von Schadcode bereits gängige Praxis [SS16, S. 228]. Allerdings bestehen hierbei einige Nachteile, wie z.B. die Flüchtigkeitseigenschaft des Arbeitsspeichers. Der Inhalt des Arbeitsspeichers unterliegt einer hohen Fluktuation, sodass viele Ereignisse zu einem späten Zeitpunkt der Analyse möglicherweise nicht mehr verfügbar sind. Eine Analyse des Prozessverhaltens über die gesamte Laufzeit bietet daher großes Potential, Ereignisse und Aktivitäten vollständig zu erfassen und unmittelbar in einen zeitlichen Zusammenhang zu setzen. Kompromittierte Prozesse oder verdächtige Aktivitäten könnten hierdurch frühzeitig, bestenfalls in Echtzeit, analysiert und erkannt werden (Live-Forensik). Auch für den Bereich des Incident Response und der Malware-Forensik, verschafft Wissen über das Verhalten von (noch unbekannter) Schadsoftware oder kompromittierten Systemen einen Vorsprung, um sich vor Angriffen mit Hilfe gezielter Gegenmaßnahmen und Erkennungsmechanismen schützen zu können.

Malware ist darauf ausgerichtet, möglichst lange unentdeckt zu bleiben und sich persistent in einem IT-System einzunisten (sog. Anti-Forensik Techniken). Dabei erstellt sie sich nicht selten einen Überblick der umgebenden Netzwerkinfrastruktur, um sich auch auf weitere IT-Systeme auszubreiten (sog. Lateral Movement).

2 Einleitung

2.1 Ausgangssituation

Windows-Betriebssysteme verfügen über ein integriertes Protokollierungssystem, das Aktivitäten des Betriebssystems zentral erfasst und für die Ablauf- und Fehlerprotokollierung von Anwendungen eingesetzt wird (Ereignisprotokollierung). Die Sammlung und Korrelation von Logdateien eignet sich grundsätzlich zu forensischen Untersuchungen [Bun11, S. 90, S. 199-201].

Mit Windows 2000 steht zusätzlich das sog. Event Tracing for Windows (ETW) als Kernkomponente zur Erfassung von Ereignissen zur Verfügung, das mit der Einführung von Windows Vista um ein sehr detailliertes Logging von Anwendungsereignissen erweitert wurde. Die Schnittstellen zur Nutzung von ETW wurden gleichzeitig für Entwickelnde und Analysierende weiträumig geschaffen und eröffnet. Fehlermeldungen und Informationen zu (Betriebssystem-)Komponenten wie Software, Treibern oder Diensten lassen sich mittels bereitgestellter ETW-Anbieter (Provider) im laufenden Betrieb dynamisch aus dem User- oder Kernelmode erheben. [PB19]

Insgesamt stehen in Windows 10 mehr als 1000 registrierte ETW-Provider zur Verfügung. Bisher wurde ETW insbesondere intern von Microsoft-Entwickelnden und Softwareherstellenden zur System- und Software diagnose, Fehlerbehebung oder Performanceanalyse eingesetzt [Mic20g]. Inzwischen sind auch einige Sicherheitsforscher/-innen auf den Nutzen und die Vorzüge von ETW aufmerksam geworden.

In der Studie zu Systemaufbau, Protokollierung, Härtung und Sicherheitsfunktionen in Windows 10 des BSI wurde u.a. dargestellt, dass ETW auch zur Erhebung von Telemetriedaten durch das Betriebssystem verwendet wird [Bun18, S. 11].

Im Vergleich zur Windows-Ereignisprotokollierung (engl. *Windows Event Log*), in der der Fokus auf einer Zusammenfassung systemrelevanter Ereignisse liegt und diese für Systemadministratoren und Anwender entsprechend aufbereitet werden, lassen sich mit ETW Abläufe und Aktivitäten ganzheitlich und detailliert verfolgen [Wol09].

Zur Nutzung von ETW ist zunächst eine gezielte Konfiguration (beispielsweise über

die ETW-API) notwendig, damit Ereignisse zur Ablaufverfolgung für eine Auswertung aufgezeichnet werden.

2.2 Zielsetzung

Die vorliegende Master-Thesis hat zum Ziel, Möglichkeiten des Einsatzes von Event-Tracing for Windows (ETW) zur Unterstützung forensischer Analysen von Prozessverhalten in Windows 10 aufzuzeigen und zu eruieren. Bisherige Forschungsergebnisse sollen hierzu konsolidiert und für die Identifizierung und Betrachtung forensisch relevanter ETW-Provider als Basis herangezogen werden. Gleichzeitig soll eine strukturierte Vorgehensweise zur Prozessanalyse entwickelt werden, mit der sich Prozesse in Windows 10 unter Zuhilfenahme von ETW analysieren lassen. Durch einen Demonstrator soll für einen konkreten Anwendungsfall praxisnah gezeigt werden, ob und inwiefern sich ETW zur Analyse von Prozessverhalten in Windows 10 eignet. Hierbei werden außerdem potenzielle Grenzen und Herausforderungen aufgezeigt und beschrieben.

2.3 Aufbau der Arbeit

Im Rahmen der Master-Thesis werden die Möglichkeiten zum Einsatz von Event Tracing for Windows (ETW) zur Unterstützung forensischer Analysen von Prozessverhalten in Windows 10 untersucht. Besonderer Fokus liegt hierbei auf Verhaltensweisen, die sich zugleich auch bei gängiger Malware beobachten lässt.

Kapitel 3 definiert und vermittelt die zugrundeliegenden Begrifflichkeiten und das erforderliche Wissen über Betriebssystemkonzepte, wie Prozesse, mögliches Prozessverhalten und zentrale (System-)Programmierschnittstellen. Die vorhandenen Logging-Funktionalitäten und ihre Zusammenhänge im Betriebssystem Windows 10 werden vorgestellt und ETW gegenübergestellt.

Der aktuelle Stand der Technik und bisherige Forschungsarbeiten zu den Einsatzmöglichkeiten und der Verwendung von ETW werden im Kapitel 4 vermittelt. Verschiedene Werkzeuge und Hilfsmittel erleichtern die Nutzung der Schnittstellen zum ETW und sollen daher innerhalb der Arbeit für einen unterstützenden Einsatz betrachtet werden. Der Einsatz und die Verwendung von ETW ist mit einigen Herausforderungen in Theorie und Praxis verbunden, die sich größtenteils bereits aufgrund der vorgegebenen Designentscheidung und der Implementierung durch

den Hersteller ergeben. In den Kapiteln 5 und 6 werden diese Herausforderungen besonders berücksichtigt.

Eine mögliche Vorgehensweise zur Analyse von Prozessverhalten unter Einsatz von ETW wird in Kapitel 5 nach einem Best Practice - Ansatz verfolgt. Hierbei stellen die Erarbeitung und Beschreibung eines prinzipiellen Ablaufs und einer Herangehensweise bei der Prozessanalyse einen zentralen Mittelpunkt dar. Wesentliche und zu berücksichtigende Eigenschaften von ETW, wie z.B. die Struktur einzelner Events oder unterschiedliche Methoden bei der Aufzeichnung von ETW-Sitzungen werden erläutert und gegenübergestellt.

Durch die Entwicklung eines Demonstrators (Proof-of-Concept) in Kapitel 6, soll die Vorgehensweise zur Analyse von Prozessverhalten aus Kapitel 5 evaluiert und in die Praxis überführt werden sowie einem konkreten Anwendungsfall (Malware-Forensik) unterzogen werden. Der Anwendungsfall wird mit Hilfe eines im Rahmen der Thesis entwickelten ETW-Prozessanalysewerkzeugs vorgenommen.

Grenzen und Herausforderungen, die sich beim Einsatz von ETW zur Unterstützung der Analyse von Prozessverhalten – insbesondere über die Kapitel 5 und 6 – ergeben haben und identifiziert wurden, werden im Kapitel 7 dargestellt und beschrieben.

Abschließend sollen die gewonnenen Erkenntnisse im Kapitel 8 einer ausführlichen Bewertung unterzogen werden. Diese soll insbesondere Aufschluss darüber geben, inwiefern der Einsatz von ETW für die unterstützende Analyse von Prozessverhalten geeignet ist und für weitere Forschung und Einsatzzwecke genutzt werden kann.

Die ausschlaggebenden Ergebnisse und Erkenntnisse aus der vorliegenden Master-Thesis werden in Kapitel 9 zusammengefasst. Ansatzpunkte für auf dieser Thesis aufbauende Forschungsarbeiten und die offen verbliebenen Aspekte und Fragestellungen werden in Ausblick gestellt.

3 Definitionen und Grundlagen

3.1 Prozesse und Threads

Prozesse sind wichtiger und zentraler Bestandteil in den Konzepten von modernen Betriebssystemen. Ein Prozess repräsentiert ein ausgeführtes Computerprogramm innerhalb eines Betriebssystems und verfügt über die erforderlichen Ressourcen, wie beispielsweise einen virtuellen Speicheradressbereich, ausführbaren Code, geöffnete Handles zu Systemobjekten und Umgebungsvariablen. Prozesse ermöglichen eine gleichzeitige Nutzung und Ausführung von Anwendungen und Programmen innerhalb eines Betriebssystems (Multitasking). Die Verwaltung der ausgeführten Prozesse erfolgt zentral durch das zugrundeliegende Betriebssystem. Im Betriebssystemkern wird eine Prozesstabelle über alle aktuell ausgeführten Prozesse geführt. Einträge in dieser Tabelle repräsentieren jeweils die Metadaten eines Prozesses und werden als Prozesskontrollblock (engl. *Process Control Block* / PCB) bezeichnet. Allen Prozessen wird durch das Betriebssysteme zudem eine eindeutige Identifikationsnummer (PID) vergeben. [Mic18b; TB15, S. 94f]

Für das Prozessmanagement wird der Lebenszyklus von Prozessen in mehrere Phasen eingeteilt. Je nach Betriebssystem kann die Anzahl der verschiedenen Prozesszustände sowie ihre exakte Bezeichnung variieren. Die prinzipiellen Hauptphasen nach dem 6-Zustands-Prozessmodell [Bau17, S. 144-149] sind:

- Neu (engl. *new*)
- Bereit (engl. *ready*)
- Rechnend (engl. *running*)
- Suspendiert (engl. *suspended*)
- Blockiert (engl. *blocked*)
- Beendet (engl. *exit*)

Zusätzlich lassen sich auch innerhalb eines Prozesses einzelne Teile (Subroutinen) des Programms über sog. Threads unterteilen und hierdurch quasi-parallel ausführen. Ein Prozess in Windows-Betriebssystemen besteht aus mindestens einem Thread, dem sog. Hauptthread (engl. *main thread*). Threads segmentieren Prozesse innerhalb

des virtuellen Speicheradressbereiches und ermöglichen eine simultane Ausführung einzelner Aktivitäten und Sequenzen eines Computerprogrammes (sog. Multithreading). Im Gegensatz zu Prozessen, gestaltet sich die Verwaltung von Threads für Programmierer jedoch leichtgängiger, da sie keinen gesonderten Adressraum im Speicher benötigen. Sie werden daher oftmals als leichtgewichtige Prozesse (engl. *lightweight process*) bezeichnet. [TB15, S. 102-106]

3.2 Eltern- und Kindprozesse

Werden durch einen Prozess weitere Prozesse erstellt, so werden diese vom Elternprozess (engl. *parent process*) erstellten Prozesse als Kindprozesse (engl. *child process*) bezeichnet. Hierdurch ergeben sich verkettete Prozesshierarchien. Ein Kindprozess hat immer nur einen Elternprozess [TB15, S. 91f]. In Windows-Betriebssystemen gibt es jedoch kein explizites Konzept, das solche Hierarchien regelt. Es ist daher möglich, einen Kindprozess vom Elternprozess zu lösen, damit dieser nach Terminierung des Elternprozesses fortbesteht. [Mic18o; TB15, S. 61f]

3.3 Prozesserzeugung unter Windows

Die Prozesserzeugung in Windows (siehe Bild 3.1) lässt sich nach [Yos+17] in sieben Phasen gliedern, die im Folgenden für einen Überblick kurz zusammengefasst werden: Vorbereitend werden in Phase 1 die übergebenen Parameter und Flags (sog. CreationFlags, [Mic20d]) der `CreateProcess*`-Funktionen konvertiert, ausgewertet und berücksichtigt. Das Betriebssystem legt für den Prozess eine Priorität fest und überprüft, ob ein Debugging erfolgen soll. Durch die Funktion `NtCreateUserProcess` werden die Argumente validiert und hinsichtlich einer schadhafte Absicht überprüft. Nach Überprüfung aller Flags wird in Phase 2 das auszuführende Windows-Abbild (engl. *executable*) geöffnet und ein Abschnittsobjekt erstellt, welches zu diesem Zeitpunkt allerdings noch nicht auf den Arbeitsspeicher abgebildet wird. In Phase 3 und 4 wird mittels der Funktion `NTCreateUserProcess` die gültige und bereits geöffnete Windows-Abbilddatei auf den Adressraum eines neuen Prozesses abgebildet und in mehreren Teilschritten ein Windows-Exekutivprozessobjekt erstellt, mit dem das Abbild ausgeführt wird. Die Prozessinitialisierung wird in Phase 5 mit der Ausführung spezifischer Operationen auf das Windows-Subsystem abgeschlossen. Nachdem die Prozessumgebung festgelegt wurde, wird in Phase 6 der ursprüngliche Thread

wieder aufgenommen, um die restlichen Prozessinitialisierungsschritte der Phase 7 im Kontext des neuen Prozesses abzuschließen. [Yos+17, S. 129-149]

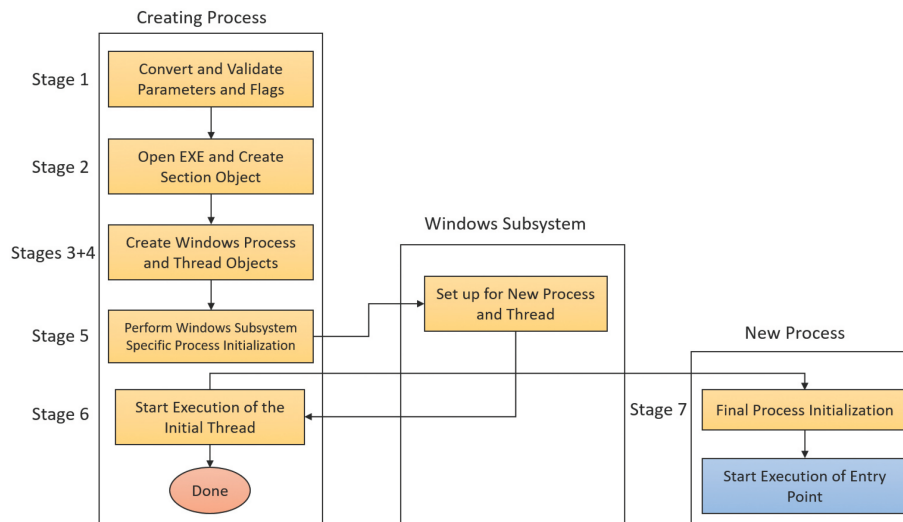


Bild 3.1: Hauptphasen der Prozesserstellung [Yos+17, S. 130]

3.4 Virtueller Speicher

Die Verwaltung des physischen Hauptspeichers (RAM) obliegt ausschließlich dem Betriebssystem (siehe Bild 3.2). Werden Prozesse erzeugt, wird ihnen durch die Speicherverwaltung des Betriebssystems ein Bereich des Arbeitsspeichers, als virtueller Speicher und Adressraum, reserviert und zur Verfügung gestellt. In diesem zugewiesenen virtuellen Speicherbereich speichert ein Prozess seine erforderlichen Daten und den ausführbaren Code. Der Zugriff auf die virtuellen Speicherbereiche anderer Prozesse oder ein direkter Zugriff auf den physikalischen Arbeitsspeicher wird durch das Betriebssystem unterbunden. [Sin15; Yos+17, S.21-23]

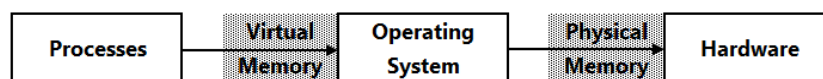


Bild 3.2: Virtueller Speicher [Sin15]

3.5 User- und Kernel-Mode

Bei der Abarbeitung von Befehlen auf dem Prozessor wird zwischen User- und Kernel-Mode unterschieden. Der Betriebssystemkern wird im Kernel-Mode mit einem Hard-

warezugriff über die sog. Hardwareabstraktionsschicht ausgeführt, während auf dem Betriebssystem ausgeführte Anwendungsprogramme der Benutzenden grundsätzlich im User-Mode ausgeführt werden. Mit (System-)Programmierschnittstellenaufrufen können ausgeführte Prozesse innerhalb des User-Mode indirekt auf die Hardware zugreifen. Der Prozessor wechselt je nach Code zur Ausführung zwischen den beiden Modi seinen Kontext. [Yos+17, S.23-25]

3.6 (System-)Programmierschnittstellen

3.6.1 Windows API (Win32-API)

Das Windows Application Programming Interface (Win32-API) ist eine für den User-Mode geöffnete Systemprogrammierschnittstelle sowohl für 32-Bit als auch 64-Bit Anwendungen. Beschrieben wird die Schnittstelle mit ihren Funktionen für Programmierer und Softwarehersteller (sog. Programming Reference) in der Dokumentation zum Windows Software Development Kit (SDK) [Mic17a]. Die Win32-API umfasst mehrere Tausend Funktionen, auf die Softwareentwickelnde über verschiedene Bibliotheken zurückgreifen können. Die Aufrufe im Programmcode werden auf native Betriebssystemfunktionen der sog. nativen API abstrahiert. [Yos+17, S.4f]

3.6.2 Windows Native API

Die native API befindet sich im Windows Kernel und wird daher auch als Kernel-API bezeichnet. Im Unterschied zur Win32-API ist die native API überwiegend für die Öffentlichkeit undokumentiert und direkte Aufrufe auf ihre Funktionen sollen aus dem User-Mode nicht möglich sein. Auf ausgewählte Funktionen der nativen API kann jedoch aus dem User-Mode über die Windows API indirekt zugegriffen werden. [Yos+17, S. 72]

3.7 Verhalten von (Malware-)Prozessen

Aktivitäten auf einem Betriebssystem lassen sich auf Prozesse zurückzuführen. Ihre Verhaltensweisen rücken daher in einen zentralen Fokus für forensische Untersuchungen. Das Verhalten von Prozessen orientiert sich an den Abläufen und Funktionen, welche durch die Softwareherstellende und Entwickelnde des zugrundeliegenden Programmcodes implementiert wurden. Allerdings können Prozesse dynamisch vom

vorgesehenen Verhalten abweichen und in nicht vorgesehene Zustände übergehen, beispielsweise durch fehlerhafte Benutzerinteraktion oder (gezielte) äußere Fremdeinwirkung, z.B. eines Angreifenden. Im Folgenden sollen gängige Verhaltensweisen von Prozessen auszugsweise vorgestellt werden und ihrem Ursprung aus bestimmten (System-)Funktionsaufrufen der Win32-API zugeordnet werden [SH12, S. 135f].

3.7.1 Erstellen von (weiteren) Prozessen und Threads

Neben einem vom Benutzenden veranlassten Start von Prozessen beispielsweise durch das Öffnen einer Anwendung lassen sich programmiertechnisch über die Funktion `CreateProcess` bzw. `CreateThread` der Win32-API durch einen bestehenden Prozess zusätzlich weitere Prozesse oder Threads erstellen [SH12, S. 147-149].

3.7.2 Ein- und Ausbinden von Bibliotheken (DLLs)

Prozesse können weitere Ressourcen und Bibliotheken, wie Dynamic Link Libraries (DLLs) einbinden (`LoadLibrary`), um auf weitere oder von mehreren Anwendungen geteilte Funktionen zuzugreifen und hierdurch beispielsweise eine DLL Injection durchzuführen [SH12, S. 254f]. In Windows-Betriebssystemen wird zwischen statisch und dynamisch verlinkten Programmbibliotheken unterschieden. Dynamische Bibliotheken können hingegen statischer Bibliotheken zur Laufzeit eines Programmes beliebig geladen und ausgegeben werden. Die Win32-API wird über die User-Level Bibliotheken bereitgestellt und muss von den entsprechenden Anwendungen, die eine entsprechende Funktion verwenden möchten, importiert werden. [SH12, S. 145-147]

3.7.3 Ein- und Ausgabeoperationen (E/A)

Die Interaktion zwischen Prozessen und klassischen Ein- und Ausgabegeräten erfolgt im Wesentlichen über die Funktion `DeviceIOControl` der Win32-API, welche die Schnittstelle zur Steuerung der Ein- und Ausgabe von Geräten (I/O Control, IOCTL) bereitstellt [SH12, S. 206f]. Anwendungen und Prozesse im User-Mode können somit mit Kernel-Mode Gerätetreibern kommunizieren. Zu den Ein- und Ausgabegeräten zählen beispielsweise sämtliche angeschlossene Laufwerke (Festplatten) und Peripherie, wie u.a. Tastaturen, Mäuse, Webcams, Mikrofone, Bildschirme, Drucker oder Scanner. IOCTLs werden durch den I/O-Manager von Windows verarbeitet. Für jede Anfrage wird ein sog. IRP (I/O Request Packet) erzeugt, das anschließend zum angefragten Geräteobjekt geroutet wird. [Sou12, S. 13f]

3.7.4 Dateisystemzugriffe

Prozesse interagieren durch Funktionsaufrufe (z.B. `ReadFile` / `WriteFile`) indirekt mit den Dateisystemen zugrundeliegender Speichermedien, wie Festplatten oder Wechselmedien. Verzeichnisse können nach Ressourcen durchsucht (`FindFirstFile` / `FindNextFile`) oder vollständig aufgelistet werden [SH12, S. 478]. Neben dem Anlegen neuer Dateien (`CreateFile`), dem Lesen und Schreiben von Dateien (`ReadFile`, `WriteFile`) können durch Prozesse allerdings auch Dateien vollständig in den Speicher geladen werden und über Speicheradressen erreichbar gemacht werden (`CreateFileMapping`) [SH12, S. 137f]. Das Erstellen von Handles zu Dateien, die mittels File Mappings in den Arbeitsspeicher geladen wurden, ist unter Malware-Entwicklern sehr beliebt, da über diese Funktion u.a. auch PE-Dateien gelesen und modifiziert werden können, um eine schadhafte Code-Ausführung zu erreichen. Mittels `FindResource` lassen sich innerhalb von PE-Dateien oder geladenen DLL-Dateien bestimmte Ressourcen, wie bspw. Zeichenketten (Strings), Konfigurationsinformationen abrufen oder weitere (malizöse) Dateien extrahieren [Sec15]. Zusätzlich zu lokalen Dateisystemzugriffen, sind weitere Zugriffe auf von anderen Systemen freigegebene Verzeichnisse im Netzwerk möglich, die häufig durch die Verwendung des Uniform Naming Convention (UNC)-Pfades erkennbar sind.

3.7.5 Windows-Registrierungsdatenbank (Registry)

Die Windows-Registry ist eine Systemdatenbank, die alle relevanten Informationen und Konfigurationen, wie Boot-Informationen, System- und Sicherheitskonfiguration, Softwareeinstellungen oder spezifische Benutzereinstellungen zentral speichert und vorhält. Die meisten Einstellungen beeinflussen die Systemperformance und das Systemverhalten. Neueinträge und Änderungen in der Registry können durch das System, den Standardbenutzer, Administratoren, aber auch durch Software und Prozesse vorgenommen werden [Yos+17, S. 32f]. Zu den möglichen Funktionsaufrufen, die durch Prozesse veranlasst werden können zählen beispielsweise das Öffnen von Registrierungsschlüsseln (`RegOpenKeyEx`), das Setzen bzw. Hinzufügen von Registrierungsschlüsselwerten (`RegSetValueEx`) oder das Auslesen bestimmter Registrierungswerte (`RegGetValue`) [SH12, S. 139f]. Für Angreifende stellt die Windows-Registry ein beliebtes Ziel dar, Informationen über den Einsatz und die Verwendung des Systems auszulesen, um weitere Aktivitäten von den hieraus gewonnen Erkenntnissen abhängig zu machen. Außerdem besteht über die Windows-Registry die Möglichkeit,

reguläre und native Werkzeuge zu verwenden, um möglichst lange unentdeckt zu bleiben. Über die Windows-Registry kann sich ein Angreifer oder eine Malware Persistenz verschaffen, beispielsweise durch Setzen eines Eintrages im Autostart-Verzeichnis. Durch Änderung bestimmter Parameter, z.B. von Systemdiensten, kann außerdem eine Privilegienerweiterung ermöglicht werden [Rot17].

3.7.6 Inter-Prozesskommunikation

In Windows-Betriebssystemen bestehen verschiedene Möglichkeiten zur Inter-Prozesskommunikation, durch die Informationen und Daten zwischen Anwendungen ausgetauscht werden können. Hierbei wird unterschieden, ob es sich um eine Kommunikation zwischen lokal ausgeführten Prozessen handelt oder sich eine Kommunikation über verteilte Systeme, z.B. in einem Netzwerk über Client-Server-Modelle, erstreckt.

Im Folgenden werden einige Varianten möglicher Inter-Prozesskommunikation unter Windows-Betriebssystemen vorgestellt:

Zwischenablage

Die Zwischenablage ermöglicht dem Benutzer das Kopieren (`SetClipboardData`) und Einfügen (`GetClipboardData`) von Informationen verschiedener Anwendungen. Grundsätzlich wird die Zwischenablage häufig von gängigen Anwendungen unterstützt und ermöglicht dem Benutzer einen Austausch von Inhalten [TB15, S. 412].

Component Object Model (COM)

Beim Component Object Model (COM) handelt es sich um einen plattformunabhängigen und objektorientierten Schnittstellenstandard, der einzelnen Komponenten innerhalb von Software einen Codeaufruf in einer weiteren Software ermöglicht. COM-Objekte innerhalb von Prozessen werden beispielsweise dazu verwendet, um über Client-Server-Modelle zu interagieren. Um Zugang zu COM-Objekten zu erhalten, werden diese über Globally Unique Identifier (GUID) und die Funktion `CoCreateInstance` oder `CoInitializeEx` aufgerufen [SH12].

Dynamic Data Exchange (DDE)

Über das Protokoll Dynamic Data Exchange (DDE) können Daten mit verschiedenen Formaten dynamisch zwischen Anwendungen auf dem lokalen System sowie zwischen Systemen über ein Netzwerk miteinander ausgetauscht werden. Eine Ausführung von Befehlen ist anwendungsübergreifend möglich, beispielsweise um andere Anwendungen zu steuern [Yos+17, S. 643].

File Mapping

Memory-Mapped Files können von Prozessen dazu verwendet werden um Informationen und Daten miteinander austauschen. Hierzu werden beliebige Dateien über die Funktionen `CreateFileMapping` und `MapViewOfFile` in den virtuellen Speicher geladen (sog. File Mapping). Prozesse können Inhalte dieser Dateien über Zeiger (Pointer) als sog. Views betrachten oder modifizieren. Auch ein Zugriff von mehreren verschiedenen Prozessen auf eine gemappte Datei ist möglich [SH12, S. 137f]. Über die Funktionen `UnmapViewOfFile` bzw. `FlushViewOfFile` können die Views geschlossen werden. Um ein Memory-Mapped File wieder freizugeben, wird der Handle über die Funktion `CloseHandle` geschlossen. [Mic18d]

Es wird zwischen zwei Arten von Memory-Mapped Files unterschieden:

- **Persistente Memory-Mapped Files**

Die gemappten Dateien sind einer Quelldatei auf einem Laufwerk zugeordnet. Sobald der letzte Prozess seine Arbeit mit der gemappten Datei abgeschlossen hat, wird als Ergebnis zurück auf das Laufwerk geschrieben. [Mic17b]

- **Nicht-persistente Memory-Mapped Files**

Die nicht-persistenten Dateien sind keiner Datei auf einem Laufwerk zugeordnet und werden daher nach Abschluss der Tätigkeiten zur Löschung freigegeben. Nicht-persistente Memory-Mapped Files eignen sich zur Interprozesskommunikation. [Mic17b]

Pipes

Pipes stellen eine häufig genutzte Form der Interprozesskommunikation in Windows-Betriebssystemen dar. Unterschieden wird zwischen anonymen Pipes (engl. *anonymous pipes*) und benannten Pipes (engl. *named pipes*), die über die Funktionen `CreatePipe` und `CreateNamedPipeA` erstellt werden. Sie dienen zur Kommunikation

zwischen Anwendungen und Prozessen, die innerhalb eines Betriebssystems ausgeführt werden, jedoch auch zur Kommunikation entfernter Prozesse über ein Netzwerk. Sie werden außerdem im Zusammenhang mit Windows Reverse Shells missbräuchlich verwendet [SH12, S. 233]. Anonyme Pipes ermöglichen ausschließlich eine Kommunikation zwischen einem Elternprozess und den zugehörigen Kindprozessen. Benannte Pipes hingegen ermöglichen auch eine Kommunikation zwischen unterschiedlichen Prozessen. [Mic18c]

(Asynchronous) Local Inter-Procedure Calls (ALPC)

Bei Local Inter-Procedure Calls handelt es sich um eine (weitestgehend undokumentierte) Möglichkeit zur paketbasierten Inter-Prozesskommunikation, die im Windows NT Kernel implementiert ist. Sie wird zur Kommunikation zwischen den internen Windows-Subsystemen verwendet, aber auch zur Kommunikation zwischen User-Mode Prozessen, einem User-Mode Prozess und einem Kernel-Mode Treiber oder zwischen Kernel-Mode Treibern untereinander [Mic07]. Mit neueren Windows-Betriebssystemen (ab Windows Vista) wurden die performanteren asynchronen LPC eingeführt [Jur10]. ALPC werden dazu verwendet, um Remote Procedure Calls (RPCs) lokal innerhalb des Betriebssystems zu transportieren [Yos+17, S. 74].

Remote Procedure Calls (RPCs)

Remote Procedure Calls basieren auf einem Client-Server-Modell und ermöglichen eine Inter-Prozesskommunikation zwischen mehreren Prozessen, die auf verschiedenen Computern ausgeführt werden. Hierdurch können insbesondere Funktionsaufrufe und Prozeduren von Anwendungen eines entfernten Systems im Netzwerk (z.B. ein Remoteserver) von einem Clientsystem ausgeführt werden. Client und Server verfügen jeweils über ihre eigenen Adressbereiche. [Mic18m]

Windows Sockets (Winsock-API)

Als protokollunabhängige Schnittstelle ermöglichen Windows Sockets (Winsock) / Windows Sockets 2 (Winsock2) eine Kommunikation über das Netzwerk oder das Internet von Client-Server-Anwendungen, unabhängig des verwendeten Netzwerkprotokolls. Wesentliche Funktionen, die durch Prozesse beim Verwenden von Sockets verwendet werden, sind nach [SH12, S. 143]:

- die Erzeugung eines Sockets (`socket`).
- das Binden eines Sockets an eine Adresse und Port (nur Server) (`bind`).
- das Abhören eines Ports auf eingehende Verbindungen (eines Servers) (`listen`).
- das Anfragen bzw. Akzeptieren einer Verbindung zu oder von einem entfernten Socket (`connect` und `accept`).
- das Senden und Empfangen von Daten über die Puffer (`send`, `recv`).
- sowie dem Schließen von Sockets (`closesocket`).

Windows Internet (WinINet-API)

Die Windows Internet API (WinINet-API) ermöglicht Anwendungen und Prozessen den Zugriff auf entfernte Ressourcen über Protokolle, wie HTTP oder FTP. Mögliche Funktionsaufrufe sind nach [SH12, S. 145] unter anderem:

- `InternetOpen`: Eine Verbindung in das Internet wird initiiert.
- `InternetOpenURL`: Es wird eine Verbindung zu einer bestimmten URL (HTTP oder FTP) hergestellt.
- `InternetReadFile`: Ähnlich wie die `ReadFile`-Funktion, lassen sich Daten aus einer zuvor heruntergeladenen Datei lesen.

3.7.7 Weitere Funktionsaufrufe

Windows Management Instrumentation (WMI)

Bei der Windows Management Instrumentation (WMI) handelt es sich um eine Schnittstelle über die Einstellungen eines Windows-Betriebssystems aus der Ferne über ein (entferntes) Netzwerk verwaltet werden können. Die Schnittstelle wird daher vorzugsweise zur automatisierten Administration über WMI-Skripte oder Anwendungen eingesetzt [Mic18p].

Windows Dienste

Hinter Windows Diensten verbergen sich – für auf längere Dauer vorgesehene – ausführbare Anwendungen, die aus Prozessen bestehen, welche primär im Hintergrund ausgeführt werden. Um neue Dienste zu erstellen wird die Funktion `CreateService` verwendet. Installierte Windows-Dienste können, neben einem manuellen Start (`StartService`), auch zum Systemstart automatisch gestartet werden [SH12, S. 153].

Über verschiedene Parameter werden u.a. Ausführungsberechtigungen festgelegt. Die Funktion `OpenSCManager` liefert als Rückgabewert ein Handle zum Service Control Manager (SCM), der eine Datenbank über die installierten Dienste repräsentiert. Durch den SCM wird die Datenbank der installierten Dienste verwaltet [Mic18n].

Windows PowerShell

Die Windows PowerShell ist eine aufgabenbasierte Befehlszeilenshell und Skriptsprache, durch die Windows-Betriebssysteme und Anwendungen verwaltet werden können. Sie lässt sich auch zur Automatisierung einsetzen [Mic18l]. Prozesse können auf die im Betriebssystem mitgelieferte PowerShell und ihre Module zurückgreifen oder Skripte aufrufen. Die PowerShell selbst wird ebenfalls als Prozess ausgeführt.

3.8 Windows Logging-Funktionalitäten

3.8.1 Windows Event Logging (Ereignisprotokollierung)

Die standardmäßige und zentrale Erfassung von Fehlern und Ereignissen, welche durch das Betriebssystem selbst, aber auch von Anwendungen verursacht werden, erfolgt in Windows-Betriebssystemen seit Windows 3.1 über die sog. Ereignisprotokollierung (engl. *event logging*). Über die Ereignisprotokollierung können eine Vielzahl von Informationen und Erkenntnisse über ein bestimmtes System, die Nutzung sowie sicherheitsrelevante Ereignisse erfasst und ausgewertet werden (siehe Tabelle 3.1). Alle Ereignisse der Ereignisprotokollierung werden entsprechend ihres Ursprungs kategorisiert und in verschiedenen Stufen (Level) repräsentiert [SS16, S. 162-167].

Tabelle 3.1: Event-Log Kategorien in Windows 10 (nach [SS16, S. 162-167f])

Kategorie	Eingeführt mit	Beschreibung
System	Windows 3.1	Fehler, Warnungen und Informationen über Windows und (System-)Dienste
Security	Windows 3.1	Informationen über erfolgreiche bzw. fehlgeschlagene Anmeldeversuche oder anderen Authentifizierungen
Application	Windows 3.1	Informationen, Fehler und Warnungen über Anwendungen, Diensten oder Treibern
Setup	Windows Vista	Informationen über (die Installation von) Windows Sicherheitsupdates, -patches, -hotfixes
Forwarded Events	Windows Vista	Ereignisse, die von anderen Computern (aus dem Netzwerk) abgerufen wurden
Application and Services	Windows Vista	Enthält eine Vielzahl von Unterkategorien zu sämtlichen Anwendungen und spezifischen Diensten

Im Hintergrund sammelt und speichert der für das Event Logging zuständige Windows-Dienst „EventLog“ entsprechend der konfigurierten Registry-Einträge festgelegte Ereignisse. Mit der Einführung von Windows Vista hat Microsoft das Event Log System konzeptionell überarbeitet und dieses an das Event Tracing for Windows (siehe Kap. 3.8.2) angebunden [nxl20]. Ereignisse, die zur Protokollierung herangezogen werden, werden aus ETW-Quellen bezogen. Außerdem wurde das neue EVTX-Format für Event Logs eingeführt [SS16, S. 163].

Windows Event Logs werden für forensische Analysen herangezogen, wenn ermittelt und nachvollzogen werden soll [SS16, S. 162]:

- was sich ereignete und wann ein bestimmtes Ereignis vorgefallen ist,
- welche Benutzenden dabei involviert waren,
- welche Systeme betroffen und involviert waren,
- auf welche Ressourcen zugegriffen worden ist.

Das Windows Event Log verfügt ausschließlich über Informationen zu Ereignissen, die standardmäßig erfasst werden oder mittels Konfiguration zur Überwachung festgelegt wurden. Über den zuständigen Dienst (Event Logging Service) kann die Granularität der zur erfassenden Informationen und Objekte konfiguriert werden [SS16, S. 161f].

3.8.2 Event Tracing for Windows (ETW)

Event Tracing for Windows (ETW) ist ein im Kernel implementiertes Logging und Diagnose Framework von Windows-Betriebssystemen (ab Windows 2000) und wird über eine API (ETW-API) gesteuert. ETW ermöglicht eine hochperformante, dynamische Ablaufverfolgung und Protokollierung von Ereignissen, die durch Prozesse von Betriebssystemkomponenten, Anwendungen, Treibern und Diensten innerhalb des User- und Kernelmode zur Laufzeit generiert werden. Hierdurch wird ein Aktivitätstracking von Betriebssystemkomponenten und Drittanbietersoftware, wie z.B. von Anwendungen oder Treibern, in Echtzeit möglich. Primärer Einsatzzweck für das Event Tracing liegt vor allem in der Softwareentwicklung. Verhaltensweisen von Anwendungen und Programmen können gezielt und umfangreich untersucht werden. Durch Einsatz von ETW können Entwickler, als eine alternative Form des Debuggings, detaillierte Ablaufprotokolle von Software anfertigen, um Fehler- und Problemursachen zu ermitteln oder die Performance ihrer entwickelten Anwendungen zu verbessern. [PB19]

3.9 ETW als Framework in Windows

3.9.1 Übersicht

Der Aufbau, die Architektur und die Funktionsweise des in Kapitel 3.8.2 eingehend vorgestellten ETW werden in Kapitel 3.9 näher beschrieben und erläutert.

ETW als Logging Framework wird im Windows-Kernel ausgeführt und ist zentral über eine API-Schnittstelle (*ETW logging API*) nutzbar, in der das Verfolgen und Protokollieren der Events gesteuert wird. Bild 3.3 gibt einen Überblick über Aufbau und Zusammenhang der Komponenten in ETW [Mic18a].

Das Framework lässt sich in vier Komponenten gliedern [Sou12, S. 417]:

Controller:

Verwaltung und Steuerung von ETW-Sessions sowie der Beziehung zwischen ETW-Providern und ETW-Sessions.

Provider:

Bereitstellung von Ereignissen (Events) bestimmter Kategorien aus Bereichen des User- und Kernel-Mode

Consumer:

Verarbeitung empfangener Events sowie Darstellung und Parsing der Events.

Sessions:

Zwischenspeichern (Puffern) von Ereignissen aus einem oder mehreren ETW-Providern.

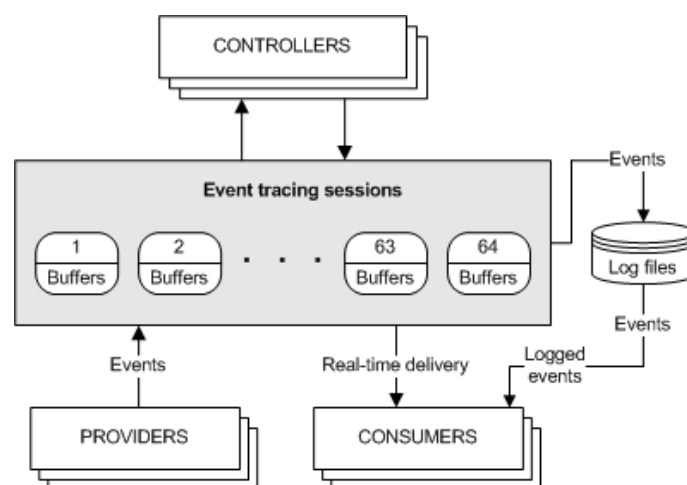


Bild 3.3: Event Tracing for Windows [Mic18a]

3.9.2 Architektur

Controller

ETW-Controller sind für die Steuerung der ETW-Sitzungen sowie die Zuordnung von ETW-Providern mit den ETW-Sessions zuständig. Sobald bestimmte Provider innerhalb einer Sitzung aktiviert wurden, empfängt die Sitzung Events, die von Providern bereitgestellt werden. Bei einem Controller handelt es sich i.d.R. um Anwendungen oder Werkzeuge, welche die ETW-Sitzungen über die ETW-API verwalten. ETW-Sessions können gestartet oder gestoppt werden und die aufzuzeichnenden ETW-Provider dynamisch zur Laufzeit festlegen. Ein Beispiel eines ETW-Controllers stellt das in Windows mitgelieferte Tool `logman` dar, mit dessen Hilfe über die Kommandozeile ETW-Sitzungen verwaltet werden können. Eine Alternative stellt das ebenfalls integrierte Windows Performance System Toolkit (WPT) dar, das über die Computerverwaltung konfiguriert werden kann. [PB19; Sou12, S. 417].

Provider

Durch die ETW-Provider, als logische Entität, werden einzelne Events aus den Bereichen des User- und Kernelmode (z.B. aus Anwendungen, Programmen, Diensten oder Treibern) erhoben und an die ETW-Sitzungen übermittelt. ETW-Provider stellen somit die Quelle der Events dar [Sou12, S. 417]. Voraussetzung hierfür ist, dass in den entsprechenden Komponenten auch Provider implementiert wurden und entsprechende Events festgelegt sind. ETW-Provider werden mittels sogenannter Event Manifeste in XML beschrieben, die Metainformationen über das verwendete Schema und Datenformat der Events sowie den Event-Deskriptoren beinhalten. Für die Verarbeitung und Interpretierung der Events aus den Providern durch die Puffer wird dieses Schema benötigt. Mittels Globally Unique Identifiers (GUIDs) werden die Event-Klassen registriert und eindeutig im System gekennzeichnet. [PB19]

Aufgezeichnete Events, die aus den einzelnen Providern zu den Sitzungen übermittelt werden, bestehen aus einem Event-Header sowie den Event-Daten, die als Array repräsentiert werden. Der Event-Header stellt Informationen über die Provider-ID (GUID) sowie den Event-Deskriptor (`EVENT_DESCRIPTOR`) bereit, der die in Tabelle 3.2 dargestellte Struktur aufweist [Sou12, S. 441f].

Tabelle 3.2: Event-Deskriptoren (nach [Sou12, S. 442])

Feld	Beschreibung
Event-ID (Id)	Eindeutiger Identifier eines Event-Typs innerhalb des Providers
Version	Versionierung von Events und zusätzlicher Identifier eines Event-Typs innerhalb des Providers
Channel	Ermöglicht eine Filterung bereits während der Analyse
Level	Granularität von Events (z.B. Fehler, Warnungen, Informationen)
Opcode	Filterung von Suboperationen
Task	Klassifiziert die Subkomponente, die das Event erhoben hat
Keyword	Klassifizierung bzw. zusätzliche Filterung von Events während der Analyse durch Schlüsselwörter

Durch die Kopfzeilen (Header) des ETW-Event-Deskriptors soll eine hierarchische Struktur der aufgezeichneten Events gegeben werden. Insbesondere fällt hierdurch eine gängige Filterung nach Keywords leichter aus. Zum Zeitpunkt des Aufzeichnens von Ereignissen durch ETW werden zusätzlich weitere Daten zum Event-Header, wie Zeitstempel, Prozess- und Thread-ID, Prozessornummer und -informationen des loggenden Threads hinzugefügt [Mic16a; Sou12, S. 441f]. Es werden folgende vier Provider-Typen nach [Mic18a; Mic18h] unterschieden, die bestimmten (Sitzungs-)Zuordnungsbeschränkungen unterliegen:

MOF Provider:

Managed Object Format (MOF) Klassen definieren Events, die von den Consumern interpretiert werden. Provider dieses Typs können nur von einer Sitzung gleichzeitig eingeschaltet werden [Mic18h].

WPP Provider:

Program database (PDB) Dateien in Binärformat enthalten die Informationen zur Dekodierung der Events. Provider dieses Typs können nur von einer Sitzung gleichzeitig eingeschaltet werden [Mic18h].

Manifest-basierte Provider:

Manifest-Dateien geben den Consumern Auskunft darüber, wie die Events strukturiert sind. Provider dieses Typs können von acht Sitzungen gleichzeitig eingeschaltet werden [Mic18h].

TraceLogging Provider:

Die Strukturen der Events enthalten alle zur Interpretierung erforderlichen Informationen und sind daher selbsterklärend. Provider dieses Typs können von acht Sitzungen gleichzeitig eingeschaltet werden. [Mic18h]

Events, die durch einen bestimmten Provider bereitgestellt werden, stammen nicht zwingend aus einer bestimmten Anwendung oder Prozess, sondern können aus mehreren Ursprungsquellen stammen. Bei der Beziehung zwischen ETW-Providern und ETW-Sessions handelt es sich um eine m:n-Beziehung [Sou12, S. 417]. Dies bedeutet, dass eine ETW-Session Events aus mehreren ETW-Providern beziehen kann und ein ETW-Provider seine Events in mehrere ETW-Sessions liefern kann.

Eine Konfiguration dieser Zuordnung beinhaltet die in Tabelle 3.3 beschriebenen Parameter und Eigenschaften:

Tabelle 3.3: Parameter zur Providerkonfiguration (nach [Gra18])

Eigenschaft	Beschreibung
Name	Providernamen (nur wenn ein Manifest existiert)
Provider	Eindeutige ID eines Providers
Level	Gewähltes Log-Level (Log Always, Critical, Error, Warning, Informational, Verbose)
KeywordsAll	Event-Kategorien können durch Angabe von Keywords gefiltert werden
KeywordsAny	Kombination von Keywords (ODER-Verknüpfung)
Properties	Optionale Eigenschaften
Filtertyp	Provider können zusätzliche Filter implementieren

Über die sog. Kernel Provider werden umfangreiche Informationen über:

- Prozess- und Threaderstellung,
- Einbindung von (System-)Bibliotheken (DLL),
- Speicherzuweisung,
- Ein- und ausgehenden Netzwerkverkehr,
- Stack Trace Accounting (sog. Sampling)

erfasst und bereitgestellt. Sie stellen daher eine besonders wertvolle Informationsquelle dar [GZF12, S. 13].

Consumer

ETW-Consumer empfangen Events aus den ETW-Sessions in Echtzeit und führen eine erste Verarbeitung, wie beispielsweise das Parsen von Daten, durch. Auch eine Betrachtung sowie die Auswertung bzw. Verarbeitung erstellter ETL-Dateien (Event Trace Log) wird mittels der Consumer vorgenommen. Speicherort und Größe der ETL-Dateien können festgelegt werden. Umgangssprachlich wird dieser Prozess daher als Konsumierung von Ereignissen bezeichnet. Häufig werden Consumer mit Controllern

innerhalb einer gemeinsamen Anwendung bzw. Werkzeugs bereitgestellt. Beispiele für ETW-Consumer sind Werkzeuge, wie der Windows Performance Analyzer (WPA), der Microsoft Network Analyzer (MMA) oder die Windows-Ereignisanzeige (Event Viewer). [Sou12, S. 417; PB19].

Sessions (Logger)

ETW-Sessions stellen den Mittelpunkt des ETW-Frameworks dar. Sie werden auch als Logger bezeichnet. Über die Event Tracing Sessions werden Ereignisse aus einem oder mehreren ETW-Providern aufgezeichnet. Dabei verwalten sie vor allem die Puffer (sog. In-Memory Buffer) und die System-Threads, welche die gepufferten Ereignisse als Aufzeichnung in der Form von sog. Event Trace Logs (ETL) - Dateien auf einen persistenten Speicher schreiben [Sou12, S. 442]. Die Inhalte von ETL-Dateien werden komprimiert und liegen daher im Binärdatenformat vor [Mic17c].

Mit den ETW-Sessions assoziierte ETW-Provider müssen nicht statisch festgelegt sein, sondern können dynamisch zur Laufzeit variieren. Die Anzahl gleichzeitig ausgeführter Sitzungen ist systemseitig auf 64 ETW-Sessions beschränkt [Mic18e]. Hiervon sind zwei Sitzungen jedoch für das System reserviert. Bestimmte Provider, die in Komponenten innerhalb des Kernel-Modus implementiert sind, können nur in speziellen Sitzungen (sog. Kernel Trace Sessions) verfolgt werden. Insgesamt können hierzu acht gleichzeitige Kernel-Sitzungen betrieben werden, von denen zwei Sitzungen bereits reserviert sind [Mic18j].

Es werden zwei Arten von ETW-Sessions unterschieden [Mic18j]:

- Real-Time Trace Sessions
- Buffered Trace Sessions

Eine besondere ETW-Session stellt die NT Kernel Logger Session dar. In dieser Sitzung werden u.a. folgende Ereignisse aus dem Kernel geliefert [Sou12, S. 422-425]:

- Erstellung von Prozessen und Threads
- Laden von Modulen
- Disk Input/Output
- File Input/Output
- Registry-Operationen
- SystemCalls
- Page Fault Events

3. Die innerhalb der Sitzungen aktivierten Provider schreiben ihre Events in die Pufferspeicher der ihnen zugeordneten Sitzungen.
4. Sofern die Ausgabe der Sitzungen in eine Event Trace Logdatei festgelegt wurde, werden die in den Puffern gesammelten Events periodisch auf die Platte geschrieben.
5. Handelt es sich um reine Echtzeit-Sitzungen, können die Puffer nur unmittelbar durch einen Consumer abgegriffen und betrachtet werden. Unabhängig davon, ob die Ausgabe der Sitzungen in eine Event Trace Logdatei vorgesehen ist, können die Pufferspeicher der einzelnen Sitzungen jederzeit auch in Echtzeit durch einen Consumer weiterverarbeitet werden.

3.9.4 Aufbau der Eventstruktur

Alle Events verfügen über einen Bezeichner, der sich i.d.R. aus der Konkatenation des Tasks mit dem Opcode (siehe Tabelle 3.2) zusammensetzt. Dem Event-Header folgt ein Array von Deskriptoren, welches die eigentlichen Daten über das Event beinhalten (Event Message). Diese werden auch als Payload bezeichnet. Die mitgelieferten Daten im Payload mit ihrer zugehörigen Struktur können individuell durch die Entwickelnden selbst bestimmt werden. Dies führt dazu, dass Daten wegen ihrer unterschiedlichen Strukturen nicht atomar vorliegen müssen. Um diese Daten aus dem Payload der Events interpretieren zu können und sie auszuwerten, ist Kenntnis und Wissen über die vorherrschenden (Daten-)Strukturen erforderlich. Auch bieten diese Informationen weitere Rückschlüsse auf Informationen, die aus den gelieferten Events entnommen werden können. Aus den im Event-Deskriptor enthaltenden Keywords (mit Beschreibung) kann eine nähere Eingrenzung der zu liefernden Events vorgenommen werden und eine Vorfilterung erfolgen. Die verfügbaren Keywords innerhalb eines Providers und deren Beschreibung eignen sich dazu, Kenntnisse über mögliche Events eines Providers zu erlangen. Da einige der Provider auch durch das EventLog verwendet werden, um Systemereignisse zu sammeln, verfügen einige Provider über einen vorformatierten Nachrichtentext mit Platzhaltern (engl. *formatted message*), die bei Erzeugung des Events mit spezifischen Informationen gefüllt werden und einen lesbaren Text bildet. Die Datenstrukturen der Events (sog. Templates) sind ebenfalls Teil der Metainformationen aus dem Providermanifest. Sie stellen eine detailliertere Informationsquelle als die Keywords dar, da aus dem Event Template unter anderem auch die Datentypen und deren zugehörigen Felder ermittelt werden können. [Sou12, S. 441-447, S. 455]

3.9.5 Abgrenzung zum Windows EventLog

Wesentliche Unterschiede zwischen Event Tracing und dem Windows EventLog ergeben sich aus der Motivation und dem verfolgten Ziel beider Ansätze bzw. Verfahren. Während ein klassisches Logging in der Regel von Systemadministratoren dazu verwendet wird, um einen zentralisierten Überblick über diskrete Ereignisse aus Anwendungen und verschiedenen Systemen ihrer Infrastruktur zu erlangen, soll beim Tracing der exakte Ablauf- und Vorgehensplan einer benutzergesteuerten Anwendung oder eines Systems, verfolgt werden. Ziel des Tracings ist insbesondere fehlerhafte Funktionen oder Störungen innerhalb des Programms- oder Systemablaufs zu erkennen, um diese im Programmcode beseitigen zu können. Event Tracing lässt sich mit einem Flugschreiber (engl. *flight recorder*) vergleichen, der detaillierte Informationen und Ereignisse in chronologischer Reihenfolge erfasst und persistent festhält. Im Vergleich zur klassischen Ereignisprotokollierung kann mittels Tracing ein per Design weitaus größerer Informationsumfang und eine höhere Informationsdichte erhoben werden. Einige der ETW-Provider lassen sich auch über die Anwendungs- und Dienstprotokolle in der Ereignisanzeige (eng. Event Viewer) konsumieren. Hierfür müssen die entsprechenden Events der Provider allerdings einem Channel zugeordnet sein, z.B. dem Analytic- oder Debug-Channel. [Rod19a; Kid19; Wol09]

4 Stand der Technik / Forschung

4.1 Einsatzmöglichkeiten von Event Tracing for Windows (ETW)

4.1.1 Debugging

Softwareentwickelnde und Programmierende können über Schnittstellen zum ETW Framework eigene Events zu ihrem Quellcode hinzufügen, um Verhaltensweisen ihrer entwickelten Programme genauer zu untersuchen. Ein wesentlicher Vorteil, der sich gegenüber eines klassischen Konsolendebuggings oder gesonderten Logdateien ergibt, ist, dass diese Debugfunktionen für einen Releasestand nicht explizit durch eine Auskommentierung innerhalb des Codes deaktiviert werden müssen. Auch etwaige Performanceeinschränkungen innerhalb der eigenen Anwendung müssen Entwickler nicht berücksichtigen, da die Verarbeitung der ETW-Events durch den Windows-Kernel abseits ihrer Anwendung erfolgt. Ein Einsatz von ETW bietet daher den Vorteil, dass sich durch die Ablaufverfolgung keine Performanceeinbrüche in den überwachten Anwendungen ergeben. Für eine Rate von 10.000 Events pro Sekunde werden bei einem 2 -Ghz Prozessor ca. 2,5 % CPU-Last verursacht. [Sou12, S. 416f]

Im Betriebssystem werden zahlreiche Ereignisse sowohl im User-Mode als auch im Kernel-Mode eigenständig bereits an vielen Stellen mitgeschnitten bzw. geloggt. Daher ist es für Entwickler nicht immer zwingend notwendig eigene Events durch ihre Anwendungen und Programme zu liefern. Es lässt sich zudem auch der Systemstart oder die Verarbeitung von Gruppenrichtlinien verfolgen. [Sou12, S. 416f]

4.1.2 Performanz- und Leistungsanalyse

Durch das Aufzeichnen und die Auswertung von Events aus Betriebssystemkomponenten sowie Drittanbietersoftware lassen sich Rückschlüsse auf mögliche Fehler- und Störquellen ziehen, um ihre Ursache zu bestimmen und zu beheben. Microsoft stellt für Entwickler mit dem Windows Performance Toolkit (WPT) als Teil des Windows Assessment and Deployment Kit (ADK) zwei wesentliche Werkzeuge zur Verfügung,

mit denen sich detaillierte Leistungsprofile von Windows-Betriebssystemen und -Anwendungen erstellen lassen [Sou12, S. 395]:

- **Windows Performance Recorder (WPR):** Aufzeichnung von Ereignissen
- **Windows Performance Analyzer (WPA):** Analysewerkzeug zur Auswertung von Aufzeichnungen

4.1.3 Windows Telemetrie

Die Funktionen von ETW werden zur Erhebung von Telemetriedaten in Windows 10 verwendet. Für die Erfassung der Diagnose- und Nutzungsdaten sind zwei ETW-Sessions zuständig: Der Diagtrack-Listener und der Autologger-DiagTrack-Listener [Bun18, S. 17f]. Letztere Sitzung wird bereits zum Bootprozess initialisiert und übergibt die gesammelten Informationen, nach Start des Windows-Dienstes „Benutzererfahrung und Telemetrie im verbundenen Modus“ (DiagTrack), an den Diagtrack-Listener. Die während des Bootvorgangs erfassten Informationen werden in zwei ETL-Dateien gespeichert [Bun18, S. 18-20]:

- `%ProgramData%\Microsoft \Diagnosis \ETLLogs\AutoLogger\AutoLogger\AutoLogger-Diagtrack-Listener.etl`
- `%ProgramData%\Microsoft\Diagnosis\ETLLogs\ShutdownLogger\AutoLogger-Diagtrack-Listener.etl`

Telemetriedaten, die in der Diagtrack-Listener Sitzung erfasst werden, werden durch diese in Form von einem Echtzeit-Protokoll-Feed an den DiagTrack-Dienst geliefert. Der Windows-Dienst DiagTrack stellt die zentrale Telemetrie-Komponente dar und ist auch für die verschlüsselte Übermittlung der erfassten Telemetriedaten an Microsoft zuständig. Durch die Ausführung des Dienstes mit SYSTEM-Rechten ist auch die Anfertigung von Speicherabbildern möglich [Bun18, S. 17; Bun20, S. 11]. Je nach konfiguriertem Telemetrie-Level in Windows 10 wird die Anzahl der einbezogenen ETW-Provider unterschiedlich festgelegt (siehe Tabelle 4.1):

Tabelle 4.1: Anzahl von ETW-Providern in ETW-Sitzungen der Windows-Telemetrie in Windows 10 1607 LTSB (nach: [Bun20, S. 10])

Telemetrie-Level	Autologger-Diagtrack-Listener	DiagTrackListener
Security	9	4
Basic	93	410
Enhanced	105	418
Full	112	422

Zu beachten ist hierbei, dass Microsoft jedoch auch über die Möglichkeit verfügt, die Anzahl der einbezogenen ETW-Provider der einzelnen Telemetrie-Level dynamisch über die Konfigurationsdatei `%ProgramData%\Microsoft\Diagnosis\Downloaded Settings\utc.app.json` festzulegen [Bun20, S. 11]. Bis auf wenige Ausnahmen werden die einbezogenen ETW-Provider lediglich über die GUID bezeichnet. Dies ermöglicht auf den ersten Blick wenig Rückschlüsse auf die Quelle der ETW-Provider. Microsoft hat über die Dokumentation zu Windows 10 veröffentlicht [Mic19a], welche Informationen mit den jeweiligen Telemetrie-Level erfasst werden. Der Informationsumfang bzw. die einbezogenen Komponenten kumulieren sich aufsteigend des festgelegten Telemetrie-Levels. Nachfolgend werden auszugsweise (Betriebssystem-)Komponenten und Bereiche zusammengefasst, aus denen Telemetriedaten erhoben werden [Mic19a]:

Security Level

- Geräte- und Betriebssysteminformationen
- Berichte aus dem Malicious Software Removal Tool (MSRT)
- Windows Defender und Endpoint Protection (z.B. Anti-Malware Signaturen, Einstellungen der Benutzerkontensteuerung (UAC), UEFI-Firmwarekonfiguration)

Basic Level

- Zusätzliche Geräteinformationen (u.a. angeschlossene Laufwerke, Anzahl Prozessorkerne, Größe des Arbeitsspeichers, IMEI)
- Qualitätsbezogene Informationen (u.a. Standby, Anzahl der Systemabstürze, Prozessor- und Arbeitsspeicherauslastung)
- Kompatibilitätsinformationen (Installierte Anwendungen und verwendete Dateien, angeschlossene Drucker und externe Speichermedien, Treiberdaten)

Enhanced Level

- Ereignisse aus Betriebssystemkomponenten (Netzwerk, Virtualisierung, Cortana, Speicher, Dateisystem)
- Berichte aus dem Malicious Software Removal Tool (MSRT)
- Speicherabsturzabbilder (Crash-Dumps, außer Heap-Dumps und Full-Dumps)

Full Level

- Informationen aus Diagnosewerkzeugen (z.B. msinfo32, powercfg oder dxdiag)
- Registrierungsschlüssel
- Speicherabsturzabbilder (inkl. Heap-Dumps und Full-Dumps)

Die von Microsoft angegebenen Informationen [Mic19a] über die von der Telemetrie erhobenen Diagnose- und Nutzungsdaten, geben ein umfangreiches Bild darüber, welche Daten aus den mit den ETW-Sitzungen assoziierten ETW-Providern der einzelnen Telemetrie-Leveln grundsätzlich gewonnen werden können.

4.1.4 Threat Hunting

Unter Threat Hunting wird der Prozess verstanden, IT-Infrastrukturen auf Anzeichen einer potenziellen Bedrohung zu untersuchen. Hierbei ergänzen sich Mensch und Maschine. Beispielsweise entscheidet ein Analyst darüber, wie von verschiedenen Sensoren erfasste Informationen zu interpretieren und einzustufen sind [LH18].

Roberto Rodriguez [Rod20] beschreibt eine mögliche Vorgehensweise, um ETW-Events aus verschiedenen Quell-Systemen mittels einer zentralen Threat-Hunting- und Analyseplattform, wie beispielsweise dem von ihm entwickelten Open-Source Framework The Hunting ELK (HELK) auszuwerten. Als Motivation zur Auswertung zusätzlicher ETW-Events neben den konfigurierbaren Windows Audit Policies mittels `auditpol.exe` führt er insbesondere an, dass diese lediglich eine Teilmenge aller in Windows-Betriebssystemen zur Verfügung stehender Telemetriedaten umfassen. Durch gezielte Analysen und Untersuchungen von ETW lassen sich weitere geeignete Datenquellen für das Threat-Hunting einbeziehen. [Rod19a; Rod19b; Rod20]

Zac Brown [Bro17a] betrachtet verschiedene Ansätze und Möglichkeiten, wie bestimmte Events zu einem verdächtigen Verhalten in Windows aus ETW-Providern herausgefiltert werden können, um sie idealerweise an ein Security Information and Event Management (SIEM) weiterzuleiten. Außerdem stellt er eine „Forensic Wishlist“ vor, die ETW-Provider aufzählt, aus denen wichtige Erkenntnisse über ein Prozessverhalten gewonnen werden können. So lassen sich beispielsweise DNS-Auflösungen über Events des Providers Microsoft-Windows-DNS-Client aufzeichnen oder Verbindungen zu Endpunkten aus dem Kernel Network Provider extrahieren. PowerShell Befehlsaufrufe und Funktionen werden durch Events des Providers Microsoft-Windows-PowerShell protokolliert. [Bro17a; Bro17b]

Alexander Bode und Niels Warnars [BW20] von der Universität Amsterdam setzen sich mit der Forschungsfrage auseinander, inwiefern .NET Agenten gängiger C2 Frameworks mit Hilfe von ETW detektiert werden können. Sie betrachten verschiedene Agents von C2 Frameworks und fokussierten sich auf das Nachladen dynamischer Komponenten in den Prozessspeicher eines .NET C2 Agents. Hierbei zeigten sie mit ETW-Events auf, dass mehrere der untersuchten C2 Frameworks zusätzliche Komponenten dynamisch zur Laufzeit nachladen. Nachweise lieferten dazu vor allem die Events `AssemblyLoad` und `ModuleLoad` des Provider `Microsoft-Windows-DotNETRuntime`. [BW20]

4.1.5 IT-Forensik

IT-Sicherheitsforscher und Forensiker haben in der Vergangenheit aufgezeigt, dass sich ETW grundsätzlich auch für einen Einsatz in der IT-Forensik eignet. Es lässt sich zwischen einem Einsatz von ETW in der Live-Forensik und der Post-Mortem Forensik unterscheiden.

Nicole Ibrahim der GC-Partners LLC befasst sich mit der Gewinnung von forensischen Artefakten aus ETW sowie den gespeicherten ETL Sessions in Form von Event Trace Log (ETL) - Dateien (Post-Mortem). Sie führt an, dass Windows-Betriebssysteme selbst standardmäßig ETW dazu verwenden, um zahlreiche systeminterne Vorgänge und Abläufe, wie beispielsweise den Shutdown- und Bootprozess oder WiFi-Aktivitäten zu verfolgen [Ibr18]. Aber auch in Microsoft Office oder der Windows Sleep-Study, in der Systemaktivitäten während des Standbys protokolliert werden, lassen sich eine Vielzahl forensischer Artefakte gewinnen. Innerhalb der analysierten ETL-Dateien können Informationen über die Nutzung von Anwendungen, Systemkonfigurationen, gelöschten Dateien, verwendete WLANs sowie physische und logische Laufwerksinformationen, wie angeschlossenen USB-Geräte, extrahiert werden [Ibr18]. Gleichzeitig weist Ibrahim darauf hin, dass nicht alle ETL-Dateien standardmäßig existieren, sodass eine Konfiguration festzulegen ist. [Ibr18]

Ben Lelonek und Nate Rogers vom CyberPoint Security Research Team [Cyb16] haben auf der Ruxcon 2016 vorgestellt, dass ein Einsatz von ETW sowohl im offensiven als auch im defensiven Sektor im Rahmen einer Live-Forensik vielversprechend ist. Neben einem Ansatz zur Erkennung von Ransomware durch einen generischen Detektionsalgorithmus auf Basis ausgewählter ETW-Events, stellen sie außerdem dar, dass aus einigen ETW-Providern, wie dem Microsoft-Windows-WinINet Provider, sensible Informationen aus Browsern und Windows Apps, wie Zugangsdaten oder Sitzungscookies, geleakt werden können. Über die ETW-Provider Microsoft-Windows-USB-UCX und Microsoft-Windows-USB-USBPORT lassen sich Eingaben von einer über USB angeschlossenen Tastatur mitloggen. [Cyb16]

Paula Januszkiewicz (CEO, CQURE Inc.) stellt in einem Blog-Post eine Möglichkeit zum Abhören verschlüsselten HTTPS-Verkehrs vor [Jan17]. Voraussetzung hierfür ist, dass Anwendungen, wie z.B. Webbrowser, die WinINet-Bibliothek des Betriebssystems verwenden. In diesem Fall lassen sich Verbindungsinhalte im Klartext aus Events des Providers Microsoft-Windows-WinInet aufzeichnen und auswerten. [Jan17]

4.2 Schnittstellen zu ETW in Windows 10

4.2.1 Computer Management - Performance (GUI)

In der Computerverwaltung (Computer Management) wird eine grafische Oberfläche zur Verwaltung der Performanz- und Leistungsmessungen in Windows 10 zur Verfügung gestellt. Hierzu lassen sich als Teil des Performance Monitors (Bild 4.1) ETW-Sessions mit einer grafischen Oberfläche erstellen und verwalten [Mic20f].

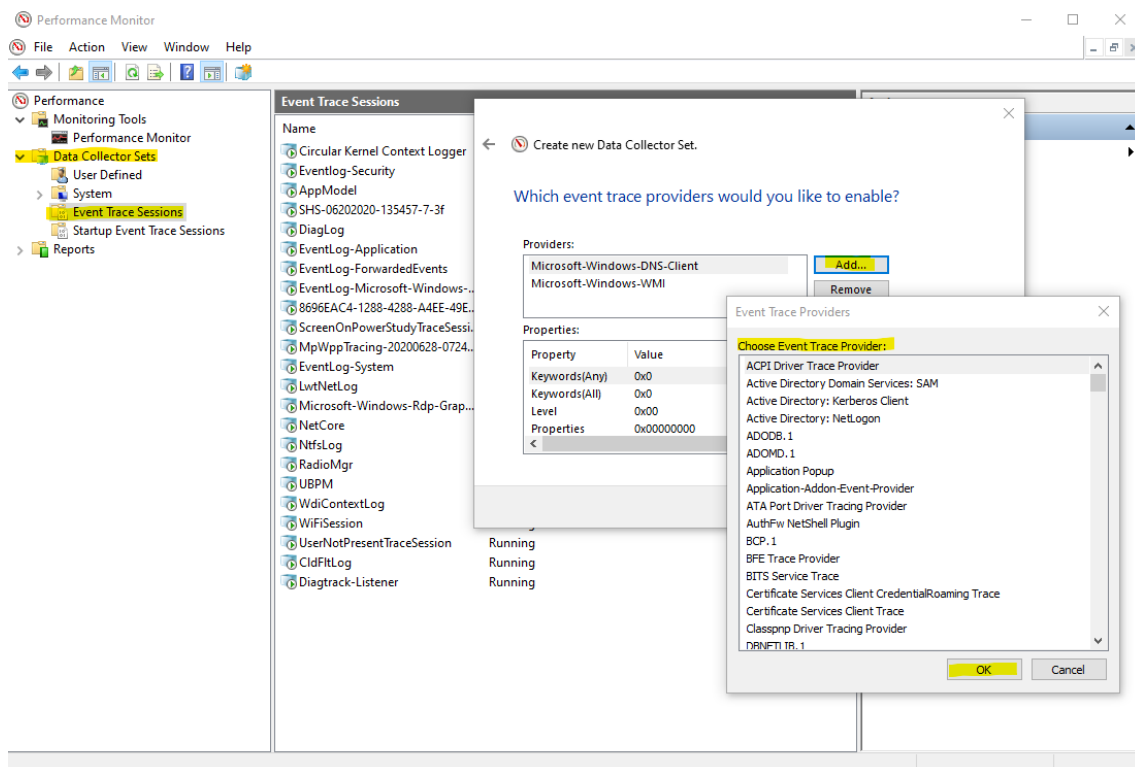


Bild 4.1: Verwalten von ETW-Sitzungen mit dem Performance Monitor

4.2.2 Logman (CLI)

Bei `logman` handelt es sich um ein Command Line Interface (CLI) zur Verwaltung der Performanz- und Leistungsmessungen über Datenkollektoren auf Basis von ETW-Sitzungen, wie sie auch über die GUI durch den Performance Monitor vorgenommen werden können [Mic18k]. Listing 4.1 zeigt einen Auszug aus der Hilfe von `logman`:

```
C:\> logman /?
Microsoft Logman.exe (10.0.18362.1)
Usage:
    logman [create|query|start|stop|delete|update|import|export] [options]
```

```

Verbs:
  create          Create a new data collector.
  query           Query data collector properties. If no name is
                  given all data collectors
                  are listed.
  start           Start an existing data collector and set the begin
                  time to manual.
[...]
  export          Export a data collector set to an XML file.
[...]
Examples:
[...]
logman create trace trace_log -nb 16 256 -bs 64 -o c:\logfile
[...]
logman start process_trace -p Microsoft-Windows-Kernel-Process -mode newfile -max
    1 -o output%d.etl -ets
logman start "NT Kernel Logger" -p "Windows Kernel Trace" (process,thread) -ets

```

Listing 4.1: Verwalten von ETW-Sitzungen mit logman

4.2.3 Windows Events Command Line Utility (CLI)

Das Windows Events Command Line Utility (`wevtutil`) wird als Betriebssystembestandteil von Windows 10 mit ausgeliefert und ermöglicht über das CLI eine Installation sowie die Abfrage von ETW-Providern inklusive den verfügbaren Event-Informationen innerhalb der Channel, sofern die Event-Templates über einen vorformatierten Informationstext verfügen [Mic17e].

4.2.4 Event Tracing API (Win32-API)

Als Teil der Win32-API stellt die Event Tracing API (ETW-API) eine Schnittstelle zum Event Tracing for Windows (ETW) für Softwareentwickelnde und Programmierende her. Die ETW-API richtet sich primär an eine Implementierung von ETW innerhalb von User-Mode Anwendungen und kann dazu verwendet werden, um:

- ETW-Sitzungen zu verwalten,
- ETW-Provider zu registrieren und bereitzustellen,
- ETW-Events in Echtzeit bzw. aus einer Logdatei zu konsumieren [Mic18i].

Für Softwarekomponenten wie beispielsweise Gerätetreiber, die im Kernel-Mode ausgeführt werden, stehen mit dem Windows Software Trace Preprocessor (WPP) sowie der ETW Kernel-Mode API gesonderte Funktionen zur Verfügung. [Mic18i]

4.3 Werkzeuge und Hilfsmittel für ETW

4.3.1 Wrapper

Bei Wrappern handelt es sich um eine zusätzliche Schnittstelle in Form einer high-level API, die die ursprüngliche, bzw. tieferliegende Schnittstelle (sog. low-level API) umhüllt. Hierdurch kann eine vereinfachte Nutzungsmöglichkeit sowie eine fehlerfreie Verwendung aufgrund reduzierter Komplexitäten geschaffen werden und für den Einsatz unter anderen Programmier- und Skriptsprachen geöffnet werden [R S20]. Nachfolgend werden einige Wrapper vorgestellt, die sich um die in 4.2.4 beschriebene ETW (low-level) API legen und Entwicklern und Sicherheitsforschenden einen vereinfachten Zugangsweg verschaffen.

Microsoft TraceEvent Library (.NET API)

Bei der Microsoft TraceEvent Library handelt es sich um einen Wrapper für die ETW-API und ermöglicht hierdurch eine vereinfachte Nutzung und Zugriff auf die Funktionen und Schnittstellen von ETW. Sie richtet sich primär an Softwareentwickler und wurde ursprünglich dazu entwickelt ETW-Events zu parsen. Über die in der Bibliothek vordefinierten Klassen können ETW-Provider innerhalb von Anwendungen aktiviert werden (`TraceEventSession`) oder ein Eventdatenstrom gelesen (`EtwTraceEventSource`) und verarbeitet werden. Die Bibliothek wird als NuGet-Package bereitgestellt und ist durch den Programmierenden einzubinden [Mic19b]. Zur Nutzung der Bibliothek wird außerdem ein umfangreicher Programmiers-Guide [Mic20e] mit unterstützenden Programmierbeispielen angeboten.

KrabsETW

Bei KrabsETW handelt es sich um eine Bibliothek für die Programmiersprache C++, die vom ehemaligen Microsoft-Entwickler Zac Brown initiiert wurde. Sie soll als Wrapper dienen, um den Einsatz und die Verwendung der ETW-API, ähnlich dem Ansatz der Microsoft TraceEvent Library, zu vereinfachen [Mic20c]. Hierzu zählt u.a. auch ein vereinfachtes Parsing der Eventdaten. Mit KrabsETW lassen sich sowohl User- als auch Kernel-Traces verfolgen. Zur Vorgehensweise der ETW-Analyse mittels KrabsETW wird von Brown folgendes Pattern angeführt, um die ETW-Daten beliebiger Anwendungen zu verarbeiten [Bro16]:

1. ETW-Sitzung erstellen
2. Einen oder mehrere Provider mit der erstellten ETW-Sitzung verknüpfen
3. Festlegen der any und all-Flags für jeden der zugeordneten Provider
4. Registrierung eines Callbacks
5. ETW-Sitzung starten
6. Erzeugung der Prozessereignisse dynamisch zur Ausführung
7. ETW-Sitzung anhalten

Zac Brown stellt zudem ein Muster zur Vorgehensweise bei der Verarbeitung der Events bereit [Bro16]:

1. Event empfangen
2. Event-Schema abrufen und auflösen
3. Event-Daten parsen (mit Hilfe des Event-Schemas)

4.3.2 Werkzeuge

In diesem Abschnitt werden einige Werkzeuge vorgestellt, die einen Einsatz und die Verwendung von ETW erleichtern und unterstützen sollen. Die einzelnen Werkzeuge können dabei nach dem beschriebenen Aufbau und der Funktionsweise von ETW verschiedene Rollen (Consumer / Controller) einnehmen.

Message Analyzer (Microsoft)

Der Microsoft Message Analyzer (MMA) ist ein von Microsoft bereitgestelltes Werkzeug, mit dem sich hauptsächlich Netzwerkverkehr analysieren lässt. Über den MMA ist darüber hinaus auch eine Echtzeitaufzeichnung von ETW-Providern möglich (Controller und Consumer). Microsoft hat den MMA zum 25. November 2019 eingestellt [Mic20a]. Eine Aufzeichnung von ETW-Sitzungen unter Einbeziehung von ETW-User und System Providern sowie das Parsing von ETW-Events ist weiterhin möglich. Langfristig ist eine Analyse darüber hinaus weiterhin mit Analysewerkzeugen von Drittanbietern, z.B. Wireshark, möglich. Logdateien im ETL-Format können hierzu mittels des Open-Source-Tools `et12pcapng` [Mic20b] konvertiert werden. Der Message Analyzer lässt sich weiterhin dazu verwenden, um geeignete ETW-Provider zu identifizieren und die Informationen einzelner Events zu untersuchen. Vorteilhaft ist die umfangreiche Filterfunktion sowie eine spezielle ETW-Ansicht (ETW-Layer).

Winshark (Wireshark Plugin)

Mit Winshark bietet das Airbus CERT ein Wireshark Plugin an, sodass sich mit Wireshark neben klassischen Netzwerkverkehr zusätzlich auch Events aus ETW über das `libpcap`-Backend aufzeichnen lassen [Air20b].

UIforETW (Google)

Das Google Open-Source Projekt UIforETW stellt eine Benutzerschnittstelle zur Aufzeichnung und Verwaltung von ETW Ablaufverfolgungen bereit [Goo18]. Google-intern wird das Projekt von den hauseigenen Entwicklern dazu verwendet, um Google-Anwendungen unter Windows, wie den Chrome-Browser, zu optimieren. Die Schnittstelle ist auch auf nicht-Programmierende ausgerichtet und stellt eine einfache Möglichkeit bereit, um Ablaufverfolgungen durchzuführen. Der Fokus von UIforETW liegt in einer erleichterten Performanceanalyse und -verbesserung [Daw15].

etl-parser (Airbus CERT)

Der Event Log File Reader (`etl-parser`) vom Airbus CERT ist eine in Python 3 geschriebene Bibliothek zum Parsen von ETL Dateien, für MOF, Manifest-basierte und TraceLogging-Provider. Für die ETW-Provider sind entsprechende Parser mitgeliefert. Es stehen folgende Möglichkeiten zur Verfügung:

- **Python-Skript `etl2xml`:** Umwandlung bekannter Events aus dem ETL-Format in das XML-Format
- **Python-Skript `etl2pcap`:** Umwandlung von Netzwerkmitschnitten mittels `netsh` in das PCAP-Format.
- **Python-Bibliothek `etl-parser`:** Python 3 Parser-Bibliothek für das ETL-Format (Linux & Windows)

Das Team vom Airbus CERT sieht in ETL-Traces hohes Potential und eine „Goldmine“ für IT-Forensiker und Incident Responder [Air20a].

TraceRpt (Microsoft)

Mit dem in Windows 10 mit ausgelieferten Werkzeug `TraceRpt` lassen sich binärkodierte Event Trace Logs (ETL), einer aufgezeichneten ETW-Sitzung in verschiedene

Datenformate (CSV, EVTX, XML) umwandeln [Mic17d]. Verwendet wird das Werkzeug über die Windows-Eingabeaufforderung als konsolenbasierte Anwendung.

pywintrace (FireEye)

Als ETW Python-Bibliothek, die von FireEye bereitgestellt wird, lässt sich `pywintrace` als flexibler Wrapper für die Windows API einsetzen, um mit Hilfe von ETW Analysen und Forschungsarbeiten unter Python durchzuführen. ETW-Sitzungen lassen sich mit `pywintrace` verwalten und steuern.

Das Paket besteht aus den Hauptklassen [Fir17]:

- **EventProvider:** Wrapper für ETW-Provider
- **EventConsumer:** Konsumieren von ETW-Sitzungen
- **ETW:** Hauptklasse für aufgezeichnete ETW-Sitzungen

Windows Events Providers Explorer (Elias Bachaalany)

Mit dem von Elias Bachaalany entwickelten Windows Events Providers Explorer in der Version v.1.2 (WEPEXplorer) wird eine grafische Schnittstelle zu den Metadaten der systemseitig registrierten ETW-Provider und der Event-Strukturen, analog zum in Windows integrierten Werkzeug `logman` bereitgestellt [Bac16; Mic16c]. Wesentlicher Vorteil besteht in der vereinfachten Bedienung und Filterung bestimmter Informationen. Gegenüber dem ETWExplorer lassen sich im WEPEXplorer auch die vorformatierten Event-Messages für ein entsprechendes Ereignis einsehen (sofern verfügbar).

ETWExplorer (Pavel Yosifovich)

Der ETWExplorer (v.0.39) von Pavel Yosifovich [Yos19a] verfügt im Wesentlichen über einen ähnlichen Funktionsumfang wie der WEPEXplorer, um Manifeste von im System registrierten ETW-Providern zu betrachten. Gegenüber dem WEPEXplorer ist es mit Hilfe des ETWExplorers möglich, die Metadaten auch im XML-Format zu exportieren. Die Event-Templates, welche Aufschluss über die Datenstrukturen innerhalb eines Events geben, werden im ETWExplorer übersichtlicher dargestellt als im WEPEXplorer (mittels PopUp-Window). [Yos19a]

4.3.3 Analysetools und Frameworks

Unter Analysetools und Frameworks werden in diesem Abschnitt ganzheitliche Konzepte und Ansätze vorgestellt, die ETW für einen bestimmten Anwendungsfall (z.B. der Prozessanalyse) instrumentalisieren. Der Fokus auf den hier vorgestellten Analysetools liegt im Einsatzbereich der IT-Forensik.

SilkETW (FireEye / Ruben Boonen)

SilkETW ist ein in C# geschriebener Wrapper für ETW, das als eigenständiges Framework betrachtet werden kann. Ziel von SilkETW ist insbesondere eine vereinfachte Verwendung von ETW, die mittels Kommandozeilenanwendung oder als Windows-Dienst erfolgen kann [Boo19a]. Innerhalb des ETW-Frameworks ist SilkETW sowohl als Controller als auch als Consumer einzuordnen. Bild 4.2 zeigt die Windows-Dienst Variante, bei der mittels Konfigurationsdatei alle Parameter, wie die zu aktivierenden ETW-Provider und zu erstellenden ETW-Sitzungen, vor Ausführung des Dienstes festgelegt werden. Bei Ausführung des SilkETW-Dienstes werden, die aus den ETW-Sitzungen in Echtzeit gelieferten Events, nach einer Verarbeitung und möglichen Filterung, z.B. mittels YARA-Rules, an das SilkService-Log weitergeleitet. Dieses kann dann beispielsweise über die Windows-Ereignisprotokollierung ausgewertet und weiterverarbeitet werden. Eine Weiterleitung besonderer Ereignisse an ein SIEM (siehe Kapitel 4.1.4 Threat Hunting) ist über diese Variante und mittels zusätzlicher Werkzeuge realisierbar. Die Implementierung von SilkETW als Windows-Dienst wurde erst nachträglich ergänzt. Über die Kommandozeilenanwendung von SilkETW lassen sich alle Operationen auch manuell und ohne Konfigurationsdatei, als Standalone-Anwendung, durchführen [Boo19a].

Mit SilkETW sollen hierdurch vor allem die beiden Hauptprobleme bei der Verwendung von ETW als Analysewerkzeug adressiert werden, bei denen es sich einerseits um das enorme Informationsvolumen und andererseits um die hohe Komplexität handelt. Mit SilkETW möchte der Entwickler Ruben Boonen von FireEye, ETW - auch abseits der Softwareentwicklung - für Untersuchungen von IT-Forensikern und Analysten zugänglich zu machen [Boo19b].

In der derzeitigen SilkETW-Version (v.0.8) lassen sich die erfassten ETW-Daten aus dem User- und Kernel-Space nur in das Windows Event Log - Format (.etvx) sowie in das JSON-Format exportieren. Ein Speichern der ETW-Daten in .etl-Dateien ist in Planung [Boo19a].

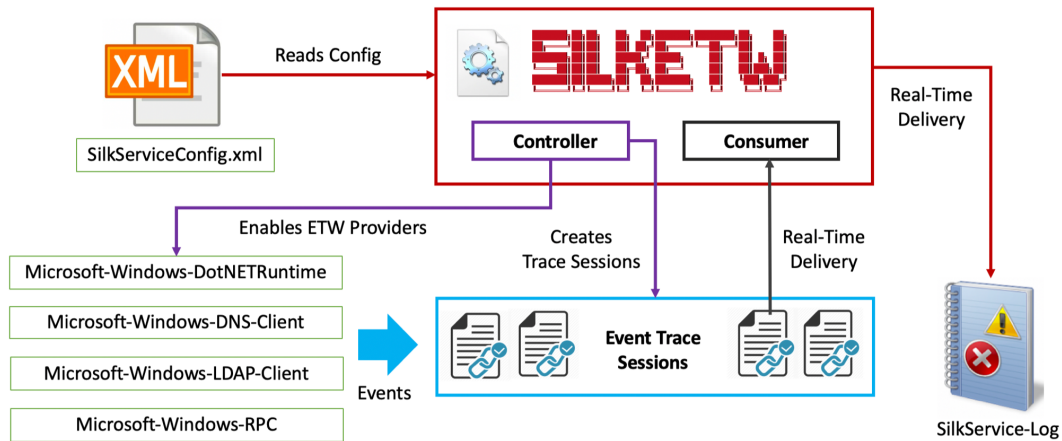


Bild 4.2: SilkETW im Betrieb als Windows-Dienst [Rod19b]

PowerKrabsETW (Zac Brown)

Das von Zac Brown entwickelte PowerKrabsETW stellt ein PowerShell Interface (API) für eine Echtzeitanalyse mittels ETW dar, das auf die Funktionen von KrabsETW (siehe 4.3.1) zurückgreift. Dabei wird auf die KrabsETW-APIs zurückgegriffen. Zu beachten ist, dass es sich um ein experimentelles Tool handelt [Bro18]. Zac Brown sieht für die Verwendung des PowerKrabsETW zwei mögliche Szenarien vor:

1. Prozessverhaltensanalyse mittels Trace-KrabsEtWProcess (ähnlich wie mit dem ProcMon-Tool der Sysinternals):
 - a) IPv4/IPv6 TCP und UDP-Daten
 - b) DNS-Namensauflösung
 - c) Remote Thread Injection
 - d) Erstellung von Kind-Prozessen
 - e) WMI-Aktivität
 - f) Registry-Aktivitäten
 - g) Datei-Operationen
 - h) PowerShell Funktionsaufrufe
 - i) (Nach-)Laden von DLL-Bibliotheken
2. Erstellung von bestimmten Providern, Filtern und Sitzungen, als flexibler Ansatz für experimentelle ETW-Analysen.

Übermittelte Events werden aus den folgenden ETW-Providern aufgezeichnet:

- Microsoft-Windows-DNS-Client
- Microsoft-Windows-Kernel-File

- Microsoft-Windows-Kernel-Network
- Microsoft-Windows-Kernel-Process
- Microsoft-Windows-Kernel-Registry
- Microsoft-Windows-PowerShell
- Microsoft-Windows-WMI-Activity

PSWasp / PSalander (Matthew Hastings)

Das PowerShell-Modul PSWasp ist auf IT-forensische Analysen ausgerichtet. Bestandteile einzelner ETW-Provider können nach bestimmten Keywords ausgewählt werden und mittels einer ETW-Sitzung aufgezeichnet werden. Die Auswertung erfolgt über integrierte Parser [Has17]. Mit dem PowerShell-Command `Start-ETWForensicCollection` wird eine ETW-Analysesitzung angelegt, die Events aus bereits ausgewählten Providern aufzeichnet und in eine ETL-Datei exportiert:

- Microsoft-Windows-Kernel-Process (Process, Thread, Image)
- Microsoft-Windows-Kernel-File (Create, Create New File, Delete-Path)
- Microsoft-Windows-Kernel-Network
- Microsoft-Windows-PowerShell
- Microsoft-Windows-DNS-Client

Zusätzlich werden Events aus der NT Kernel Logger Sitzung aufgezeichnet, die auf die Keywords Process und Thread reduziert werden. Mit `Get-ETWForensicEventLog` lassen sich die Events aus der erzeugten ETL-Analysesitzung parsen und in das XML-Format (z.B. mit `Export-Clixml`) exportieren. Anschließend Ereignisse aus der XML-Datei in der PowerShell filtern.

Extended Process Monitor (Pavel Yosifovich)

Bei ProcMonX (v.022 Beta) [Yos19b] handelt es sich nach Pavel Yosifovich um eine alternative Variante des Process Monitors (ProcMon) von Mark Russinovich [Rus19], die anstelle von Minifiltern (über Kernel-Mode Treiber) und Process/Thread-Callbacks, lediglich auf verfügbare Informationen und der Datenbasis aus ETW zurückgreift. Dabei wird die in Kapitel 4.3.1 vorgestellte Trace Event Library verwendet. Über eine GUI lassen sich diverse Filter setzen und Kategorien (z.B. Prozesse, Dienste, Registry, Dateisystem, Arbeitsspeicher oder Netzwerkschnittstellen) auswählen, aus denen die Events ausgewählt werden. Als Informationsquelle dient ausschließlich die NT Kernel Logger Sitzung, für die im Quellcode die Keywords „All“ für den Kernel Trace Event

Parser der TraceEvent Library festgelegt sind. User-Mode Provider werden nicht in das Tool einbezogen, sodass beispielsweise DNS-Auflösungen oder Schnittstellen von Bibliotheken wie WinINet oder WinHTTP nicht miterfasst werden.

ETWProcessTracer (Martin Reuter)

Für die Durchführung des Proof-of-Concepts in Kapitel 6 wird im Rahmen der vorliegenden Arbeit das Analysewerkzeug ETWProcessTracer als Demonstrator entwickelt und verwendet. Durch die Eigenentwicklung wird das Ziel verfolgt, eine hohe Skalierbarkeit zu erreichen. Analysen sind nicht auf eine vordefinierte Auswahl bestimmter ETW-Provider beschränkt. Das Werkzeug wird als .NET-Framework C# Konsolenanwendung unter Einbindung der TraceEvent Bibliothek (siehe 4.3.1) realisiert. Der vollständige Quelltext ist im Anhang A.1 gelistet.

Mit dem ETWProcessTracer, als analytisches Forschungswerkzeug für den Einsatz in Kapitel 6, wird nachfolgender Funktionsumfang erfüllt:

- **Erstellen einer ETW-Analysesitzung**
 - Name: „ETWProcessTracerSession“
 - Datenstrom: Echtzeitsitzung (Real Time)
- **Aufzeichnung von Kernel-Events (Kernel-Mode)**
 - Aktivieren der NT Kernel Logger Sitzung
 - Zuordnen der Kernel-Mode Provider mittels Kernel-Keywords
 - Filtern der Events nach Prozesszugehörigkeit (nach PIDs)
- **Aufzeichnung von Events registrierter ETW-Providern (User-Mode)**
 - Zuordnen von Providern zur Sitzung (alle im System registrierte Provider oder eine individuelle Auswahl über eine Konfigurationsdatei)
 - Filterung der Events nach Prozesszugehörigkeit (nach PIDs)
 - Überwachung erzeugter Kindprozesse der überwachten Prozesse (Watchlist)
 - Überwachung erzeugter Kindprozesse von zusätzlichen (System-)Prozessen (Childprocess-„SpawnDetection“)
 - Überwachung neu erzeugter Prozesse nach Prozessnamen (ProcessName, z.B. powershell, msieexec)
- **Ausgabe von Zwischenergebnissen**
 - Prozesserzeugungen / Prozessterminierung
 - Anzahl der bereits aufgezeichneten Events

• **Export der prozessbezogenen Ereignisse**

- Protokolldatei im CSV-Format:
ID, Time, PID, Provider, EventName, EventMessage

Bild 4.3 stellt die prinzipielle Funktionsweise mit Steuerungs- und Datenflüssen des ETWProcessTracers dar:

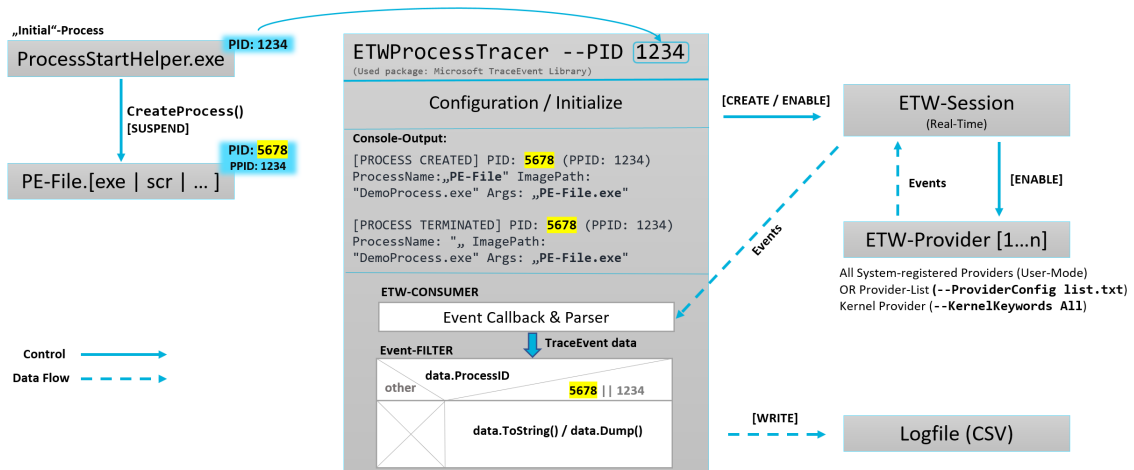


Bild 4.3: ETWProcessTracer (vereinfachte Darstellung der Funktionsweise)

Gegenüberstellung der ETW-Analysewerkzeuge und -tools

Die in Kapitel 4.3.3 vorgestellten ETW Analysetools und Frameworks werden im Folgenden gegenübergestellt (siehe Tabelle 4.2) und hinsichtlich ihres Einsatzes zur Prozessverhaltensanalyse mit ETW bewertet.

Tabelle 4.2: Gegenüberstellung der ETW-Analysewerkzeuge

Werkzeug	API	ETW-Provider (User-Mode)	Kernel Provider	Filter	Schnittstelle	Ausgabe
SilkETW	EventTracing Library (.NET)	beliebig (dynamisch)	Ja	EventName ProcessID ProcessName Opcode	CLI (Service)	JSON EventLog ELK (URL)
PowerKrabsETW	Krabs ETW (ETW-API)	7 (statisch)	Nein	ProcessName	PowerShell	PS-Objekt
PSWasp	EventTracing Library (.NET)	5 (statisch)	Ja (Keywords: Process, Thread)	Keywords EventName	PowerShell	ETL (XML)
ProcMonX	EventTracing Library (.NET)	-keine- (statisch)	Ja (Keywords: All)	ProcessName EventCategory	GUI	CSV
ETWProcessTracer	EventTracing Library (.NET)	beliebig (dynamisch)	Ja	ProcessID ProcessName	CLI	CSV

SilkETW (v.0.8) ist als Framework primär auf die forschungsbasierte Datensammlung aus ETW ausgerichtet, um diese zur Unterstützung in der Verteidigung von Angriffen einzusetzen. Während über das CLI immer nur ein einzelner ETW-Provider aufzeichnet werden kann, sind mit der Ausführung als Windows-Dienst (SilkService) über eine Konfigurationsdatei die Aufzeichnung mehrerer ETW-Provider gleichzeitig möglich. Aufgezeichnete Events enthalten den vollständigen Formatted Message Text (siehe 3.9.4). Sollen viele Provider aufgezeichnet werden, ergibt sich ein hoher Konfigurationsaufwand. Insbesondere die Weiterleitung der Events und das Windows EventLog und die Möglichkeit Yara-Rules anzuwenden, ermöglicht einen Einsatz zur Detektion von vorgegebenen Angriffsszenarien. Eine Filterung über die CLI ist immer nur auf einen Prozessnamen bzw. Prozess-ID möglich, sodass Events über erstellte Kindprozesse nicht mit aufgezeichnet werden.

Um ein ganzheitliches Verhaltensbild zu erfassen, müssen daher Provider ohne eine Filterung aufgezeichnet werden und im Anschluss gefiltert werden. Hierbei entstehen je nach Anzahl einbezogener Provider umfangreiche Datenmengen.

PowerKrabsETW (v.0.1) lässt sich neben einer Prozessverhaltensanalyse (TraceKrabsEtwProcess) als experimentelles Werkzeug auch einsetzen, um prozessunabhängige Events aufzuzeichnen. Bei der Prozessverhaltensanalyse wird ein Prozess durch PowerKrabsETW gestartet und Events aus einer vordefinierten und nicht anpassbaren Auswahl von ETW-Providern aufgezeichnet. Hierdurch wird ein guter Überblick über bestimmte Ereignisse, wie z.B. Datei- oder Netzwerkoperationen gegeben. Events aus dem Kernel Provider werden hierbei nicht herangezogen.

Durch die statische Vorgabe der ETW-Provider, können keine zusätzlichen weitere Provider ausgewählt werden, um bspw. auch Registry-Ereignisse zu erfassen. In der zweiten Variante von PowerKrabsETW lässt sich eine individuelle Aufzeichnung der vorgegeben Provider vornehmen, ohne diese auf einen Prozess zu begrenzen. Pro Provider wird in diesem Fall eine eigenständige ETW-Sitzung angelegt. Werden diese nach Abschluss der Analyse nicht ordnungsgemäß beendet, kann die maximale Begrenzung auf 64 gleichzeitige Sitzungen schnell erreicht werden.

PSWasp (v.0.0.0.1) ist auf Analysen mit forensischer Datensammlung aus ETW ausgerichtet. Die Aufzeichnung lässt sich nicht auf einen zu untersuchenden Prozess beschränken. Eine Filterung ist daher aus den aufgezeichneten Events vorzunehmen. Die Provider sind wie bei PowerKrabsETW fest vorgegeben und beziehen sich daher lediglich auf eine beschränkte Auswahl von ETW-Providern. Im Gegensatz zu PowerKrabsETW werden jedoch auch Events aus dem Kernel Provider mit aufgezeichnet.

ProcMonX (v.0.21 Beta) ist auf das Prozess-Monitoring, ähnlich dem Sysinternals-Tool ProcMon, ausgerichtet. Die grafische Oberfläche lässt sich intuitiv bedienen. Der Funktionsumfang von ProcMonX nicht mit den bereitgestellten Funktionen des klassischen ProcMon-Tools vergleichbar.

Aufgezeichnete Events lassen sich direkt auf der GUI betrachten und Filtern (z.B. nach Prozessname oder Ereignisse). Diese Funktionen in ProcMonX sind noch nicht vollständig implementiert. So lässt sich beispielsweise kein Filter auf eine PID oder die Aufzeichnung lediglich prozessbezogener Ereignisse begrenzen. Exportierte CSV-Dateien sind je nach ausgewählten Kategorien und Aufzeichnungsdauer sehr groß. Prozessbezogene Events müssen im Nachhinein aufwendig extrahiert werden. Im Export befinden sich eine Auswahl festgelegter Datenfelder der ETW-Events.

ETWProcessTracer (v.0.1) zeichnet prozessbezogene Events aus beliebigen ETW-Providern (User-Mode / Kernel-Provider) auf, damit sich forensisch relevante ETW-Provider und Events identifizieren lassen. Der Fokus liegt beim Einsatz in der Malware-Forensik. Einige Einschränkungen der vorgestellten Tools sollen durch den ETWProcessTracer ausgeglichen werden, um ein ganzheitliches Verhaltensbild über einen Prozess inklusive erzeugter Kindprozesse zu generieren. Zusätzlich können bestimmte Prozessnamen festgelegt werden, deren Prozess-ID nach Prozesserzeugung in die Analyse mitaufgenommen werden.

Zur Identifikation forensisch relevanter Provider und Events können nach dem Maximalprinzip alle beziehbaren prozessbezogenen Events aus den ETW-Providern und der NT Kernel Logger Sitzung aufgezeichnet und nach Abschluss der Analyse in eine CSV-Datei exportiert werden. Die Aktivierung aller verfügbaren ETW-Provider in der Analysesitzung bietet den Vorteil, dass sich im Rahmen der Auswertung die tatsächlich involvierten Provider mit prozessbezogenen Ereignissen ermitteln lassen. Durch das Maximalprinzip besteht das Risiko, dass eine hohe Anzahl forensisch nicht relevanter Events generiert und aufgezeichnet werden, welche die Performance des Analysesystems negativ beeinträchtigen und eine Auswertung erschwert. Für einen Einsatz in der Malware-Forensik empfiehlt sich die Einbeziehung forensisch relevanter ETW-Provider.

4.4 Herausforderungen

4.4.1 Identifikation von ETW-Providern

Mit der Einführung von Windows Vista ist die Anzahl der im Betriebssystem implementierten ETW-Provider stark angestiegen. Mit Windows 10 stehen mehr als 1000 registrierte ETW-Provider zur Verfügung. Tendenziell steigt die Anzahl der ETW-Provider und Events mit jedem weiteren Windows 10 Release [jdu20]. Zusätzlich können Softwareentwickelnde darüber hinaus selbst-definierte Provider in ihre Anwendungen und Programme implementieren, um zusätzlich anwendungsspezifische Events zu liefern. In diesem Fall obliegt es in der Entscheidung und Verantwortung eines Softwareherstellenden bzw. des Entwickelnden, ob die Manifeste der implementierten Provider mit bereitgestellt werden [Sou12, S. 426f]. Die (Daten-)Strukturen der Events können, wie auch bei den im System vorhandenen ETW-Providern, individuell durch den Programmierenden festgelegt sein [Sou12, S. 441f]. Systemintern werden Provider über die GUID adressiert. Zu einem überwiegenden Teil, der bereits im System registrierten ETW-Provider, lässt sich aus den zugeordneten Bezeichnungen (Providernamen) die Kategorie der Datenquelle herleiten. [Sou12, S. 426f]

Durch Microsoft wird für Windows 10 keine detaillierte Dokumentation zu den bereits im System registrierten ETW-Providern bereitgestellt, die Auskunft darüber geben könnte, welche möglichen Informationen über Events aus den einzelnen Providern geliefert werden. Providerbezeichner und eine Auswertung des zum Provider zugehörigen Manifests, mit den unter dem Kapitel 4.3.2 aufgeführten Werkzeugen, können erste Anhaltspunkte und Ergebnisse über die zu erwartenden Events eines Providers liefern. Informationsgehalt sowie Inhalte von Events sind vorab jedoch nicht bestimmbar. Für die im System registrierten Provider gibt es häufig sogenannte Templates, die vordefinierte Event-Messages mit Platzhaltern beinhalten. Einige Sicherheitsforscher haben in Forschungsarbeiten und Projekten bereits ETW-Provider identifizieren können, die forensisch relevante Informationen liefern (Kapitel 4.1.5).

Neben den klassischen ETW-Providern existieren in Windows 10 zusätzliche Provider des Windows software trace preprocessor (WPP) und des TraceLoggings, die in Kapitel 3.9.2 bereits eingehend erläutert wurden. Sie sind weniger bekannt als die ETW-Provider, bieten jedoch nach Aussage von Matt Graeber „[...] a potential goldmine of valuable information that has been right under your nose [...]“ [Gra19]. Sie sind in der Regel daran zu erkennen, dass sie lediglich über eine GUID verfügen und kein entsprechendes Manifest im System registriert ist. Sowohl TraceLogging

als insbesondere das WPP sind für den ausschließlichen Einsatz als Debugging vorgesehen und entworfen worden. Eine Aufzeichnung von Events dieser Provider zu anderen Zwecken ist daher standardmäßig nicht vorgesehen. Werden die Manifeste beispielsweise nicht über die PDB-Symbole mit ausgeliefert, bleibt häufig nur die Möglichkeit eines Reverse Engineering der Binärdateien von Programmen, um an Informationen zu den Manifesten dieser Provider zu gelangen und diese zu rekonstruieren. Die zugehörigen Manifeste mit vordefiniertem Schema liegen im sog. Trace Message Format (TMF) vor. [Gra19]

Für den jeweiligen Zweck und das Ziel einer Analyse mit Hilfe von ETW, kann die Menge der einzubeziehenden ETW-Provider von hoher Bedeutung sein. Mit jedem zusätzlich einbezogenen Provider werden eine Vielzahl weiterer Events geliefert. Überflüssig erfasste Informationen erhöhen jedoch die erfasste Datenmenge und können sich negativ auf die Systemperformance und -ressourcen auswirken. Die Entwickler von KrabsETW merken in den Technical Notes an, dass in einem ihrer Tests mit neun zugeordneten ETW-Providern einer ETW-Sitzung auf einem stark ausgelasteten System bereits 1,5 Trillionen Events pro Tag erfasst wurden [Bro16]. Ein hohes Risiko, das aus der Lieferung zu vieler Ereignisse ausgeht, ist zudem, dass der Thread, welcher mit der Verarbeitung der ETW-Sitzung befasst ist, nicht mehr mit dem Parsen der Eventinformationen nachkommt und Events verloren gehen können. [Bro16]

4.4.2 Konsumierung von ETW-Events

Wesentliche Aufgabe des Konsumierens von ETW-Events besteht darin, die über die Events bereitgestellten Informationen und Daten zu dekodieren und zu parsen, um sie für die spätere Auswertung aufzubereiten. Aus einem ETW-Provider können eine Vielzahl verschiedener Events mit unterschiedlichen (Daten-)Strukturen und Inhalten ausgeliefert werden. Im Gegensatz zum ausführlich dokumentierten EVTX-Log Format ist das ETL-Format nur geringfügig dokumentiert [Air20a]. In der Regel ist die eigentliche Information (sog. Payload) der Events aus den systemeigenen ETW-Providern nicht einheitlich strukturiert, was eine Aggregation der Daten erschwert. Eine (automatisierte) Verarbeitung und Filterung ist daher maßgeblich vom verfügbaren Wissen und den bereits bereitgestellten Informationen über die verwendeten Strukturen und Datentypen abhängig [Sou12, S. 426f]. Um an die notwendigen Informationen zu gelangen, eignen sich die in Kapitel 4.3.2 beschriebenen Werkzeuge. Einige Forschungsarbeiten und Werkzeuge haben sich bereits intensiv und

umfangreich mit den Datenstrukturen der Events auseinandergesetzt. Der etl-parser 4.3.2 vom Airbus CERT verfügt über zahlreiche Informationen zu den Strukturen einzelner Events aus ETW-Providern.

Innerhalb der Microsoft TraceEvent Library werden Parser bereitgestellt, die mit Hilfe der Providermanifeste die Events interpretieren lassen [Mic20e].

Wie in Abschnitt 4.4.1 bereits erläutert, steigt die Anzahl der Events zunehmend mit der Anzahl einbezogener ETW-Provider. Dies kann im schlimmsten Fall zu einer vollständigen Systemauslastung führen. Im Programmierguide der Microsoft TraceEvent Library wird angegeben, dass das Aufzeichnen von 10.000 Events pro Sekunde ein System bereits zu ca. 5 % auslastet. Eine höhere Aufzeichnungsrate von Events sollte daher vermieden werden [Mic20e].

Eine zusätzliche Problematik stellen verlorene Events dar. Sie lassen sich auf verschiedene Ursachen zurückführen. Neben fehlerhafter Konfiguration von ETW können nach Microsoft [Mic18a] folgende Gründe vorliegen:

- Events überschreiten die maximale Größe von 64 Kb.
- Der ETW Puffer ist kleiner als die Event-Gesamtgröße.
- In der Echtzeitaufzeichnung werden die Events nicht schnell genug konsumiert.
- Bei Aufzeichnung in eine ETL-Datei ist die Festplatte zu langsam, um mit der Aufzeichnungsrate mitzuhalten.

Ein Teil dieser Probleme können ausschließlich durch die verantwortlichen Softwareentwickelnden der Anwendungen und Programme gelöst werden [Mic18a].

4.4.3 Auswertung und Bewertung der Informationen

Wie in Abschnitt 4.4.1 sowie 4.4.2 bereits angeführt, lässt sich über die Auswahl der einzubeziehenden ETW-Provider und der hiermit verbundenen Anzahl gelieferter Events auch die zu erhebende Datenmenge steuern. Ziel sollte es sein, einen geeigneten Mittelweg zu finden, um einerseits keine relevanten Informationen zu verlieren und andererseits eine Vollausslastung bzw. Überlastung der Systemressourcen zu verhindern. ETW ist als Framework für einen Einsatz im Bereich des Debuggings und der Performanz- und Leistungsanalyse vorgesehen und entworfen worden. Viele der bereitgestellten Informationen beziehen sich daher auf sehr detaillierte Messgrößen und -werte. Für eine Auswertung und Bewertung für IT-forensische Fragestellungen sind diese häufig von nachrangiger Bedeutung, wenn aussagekräftige Informationen über ein bestimmtes Prozessverhalten gewonnen werden sollen.

Gespeicherte ETW-Sitzungen in Form von ETL-Dateien sind binärcodiert, was bedeutet, dass Informationen zur Dekodierung erforderlich sind. Unter Umständen besteht im Fall einer Post-Mortem-Forensik keine Möglichkeit diese Informationen nachträglich aus dem Ursprungssystem zu gewinnen. Ibrahim [Ibr18] führt hierzu an, dass bei der Registrierung eines Event Providers auch die zur Dekodierung benötigten Informationen hinterlegt werden. Folglich können Informationen aus ETL-Dateien, die von einem System mit einem zum Analysesystem abweichender Konfiguration stammen, nicht dekodiert werden. Die angestrebte vollständige Auswertung aufgezeichneter ETW-Events ist daher nur auf dem System gewährleistet, auf dem die Daten erhoben wurden. Alternativ können die Manifeste auch auf weiteren Systemen installiert und registriert werden. [Ibr18]

5 Vorgehensweise zur Analyse von Prozessverhalten

5.1 Vorbereitung und Fokus der Analyse

Aus IT-forensischer Sicht ist es bei der Analyse von Prozessverhalten von Interesse, wie sich bestimmte Verhaltensweisen und Aktivitäten von Prozessen durch den Einsatz von ETW, für eine spätere Bewertung, beobachten und erfassen lassen. Grundsätzlich können verschiedene Herangehensweisen verfolgt werden. So kann zum einen der Fokus auf die Aufzeichnung aller verfügbaren und prozessbezogenen Ereignisse gelegt werden und andererseits die gezielte Erfassung bestimmter (unerwünschter) Verhaltensweisen. Je nach gewähltem Ansatz fallen unterschiedliche hohe Datenmengen aus der Analyse an. Ein wesentliches Ziel sollte daher sein, die Datenmengen auf Grundlage gesammelter Erkenntnisse so zu reduzieren, dass sie sich auf den zu untersuchenden Prozess beschränken. Bei der Untersuchung von (noch) unbekanntem Prozessen empfiehlt es sich zunächst ein breites Verhaltensspektrum zu erfassen, das sich neben den bereits erwähnten Funktionsaufrufen der Systemprogrammierschnittstellen idealerweise auch auf Dateisystem-, Registry- und Netzwerkinteraktionen erstreckt.

Im Rahmen der Vorbereitung sollte die Durchführung anhand einer methodischen Vorgehensweise geplant werden, um eine spätere Nachvollziehbarkeit und Reproduzierbarkeit der Ergebnisse zu gewährleisten. Die hier beschriebene Vorgehensweise soll in Anlehnung an das Vorgehensmodell des forensischen Prozesses aus dem Leitfaden IT-Forensik des BSI erfolgen [Bun11, S. 59-65]. Bild 5.1 stellt das in diesem Kapitel erläuterte forensische Vorgehensmodell bei der Analyse von Prozessverhalten mit ETW und den Ablauf der einzelnen Phasen dar.

Der prinzipielle Ablauf der Analyse von Prozessverhalten kann sich inhaltlich darin unterscheiden, ob der zu untersuchende Prozess bereits auf einem System ausgeführt wird (z.B. Analyse im Rahmen des Incident Responses) oder das zugehörige Executable zum ausgeführten Prozess vorliegt und gezielt in einer Analyseumgebung ausgeführt werden kann (z.B. in der Malware-Forensik).

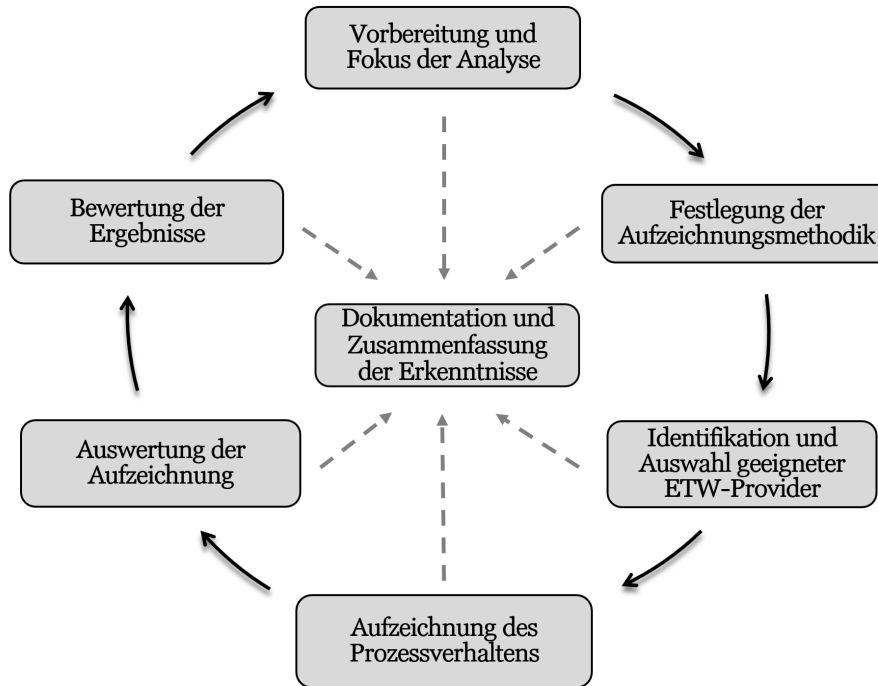


Bild 5.1: Forensisches Vorgehensmodell bei der Analyse von Prozessverhalten mit ETW

In Vorbereitung auf die Durchführung einer Analyse von Prozessverhalten mittels ETW ist zunächst der Fokus auf bestimmte Verhaltensweisen festzulegen, der sich aus dem zu untersuchenden Prozess beobachten lässt. Ein allgemeiner Überblick zu möglichen Verhaltensweisen von Prozessen wird in Kapitel 3 beschrieben. Um aus den Analyseergebnissen im späteren Verlauf der Untersuchung erkennen und bewerten zu können, ob es sich bei einer bestimmten Prozessaktivität um ein legitimes oder schadhafes und unerwünschtes Verhalten handelt, müssen die Verhaltensweisen vorab voneinander abgegrenzt werden. Wie in Kapitel 3.7 beschrieben, resultieren verschiedene Verhaltensweisen primär durch die im Quellcode eines Programmes verankerten Funktionsaufrufen der Systemprogrammierschnittstellen (APIs) des Betriebssystems. Ein maliziöser Prozess einer Schadsoftware bedient sich – konzeptionell bedingt – den gleichen verfügbaren Schnittstellen des zugrundeliegenden Betriebssystems, wie legitime Software [SH12, S. 313]. Beispielsweise können Dateien aus weiteren Quellen, wie dem Internet, nachgeladen werden, neue Prozesse und Threads erstellt werden, Registrierungswerte abgefragt und geschrieben werden oder Dateisystemoperationen durchgeführt werden.

Liegt das Ziel der Analyse in der Untersuchung des Verhaltens eines maliziösen Prozesses, sind daher, neben den für eine Malware üblichen Funktionsaufrufen, auch Informationen und Details über das Ziel eines Aufrufes zu ermitteln, um hierdurch

eine unerwünschte Aktivität von einer legitimen Nutzung der Schnittstellen zu differenzieren und abzugrenzen.

Die MITRE ATT&CK Enterprise Matrix für Windows (Adversarial Tactics, Techniques, and Common Knowledge) [MIT20b] ist eine tabellarische Übersicht über bekannte und häufig von Angreifern verwendete Taktiken und Techniken. Einige der beschriebenen Techniken basieren auf eigenständigen (maliziösen) Prozessen, andere wiederum bedienen sich bereits vorhandener (Betriebssystem-)Komponenten. Die in der MITRE ATT&CK aufgeführten Taktiken lassen sich zur Bestimmung von Prozessverhalten heranziehen und können dabei helfen, legitime Verhaltensweisen von schadhaften und böswilligen Absichten zu differenzieren.

Die Taktiken richten sich nach der Absicht eines Angreifers und erstrecken sich über die nachfolgenden Kategorien [MIT20b]:

- Initial Access (Erstzugriff)
- Execution (Ausführung)
- Persistence (Persistenz)
- Privilege Escalation (Rechteauserweiterung)
- Defense Evasion (Umgehung von Abwehrmaßnahmen)
- Credential Access (Zugriff auf Zugangsdaten)
- Discovery (Ausspähen und Entdecken)
- Lateral Movement (Laterale Ausbreitung)
- Collection (Sammlung von Daten)
- Command and Control (Befehlen und Steuern)
- Exfiltration (Ausschleusung von Daten)
- Impact (Auswirkungen auf Dienste und Prozesse)

Zur Durchführung der Analyse sollte ein gesondertes Analysesystem eingerichtet und konfiguriert werden. Dieses kann auch innerhalb einer virtualisierten Instanz ausgeführt werden. Neben der Betriebssysteminstallation von Windows 10 sind im Bedarfsfall zusätzliche Voraussetzungen und Abhängigkeiten, wie Skript- und Laufzeitumgebungen, zu schaffen, um eine kontrollierte und ordnungsgemäße Ausführung des Zielprozesses zu gewährleisten und das gesamte Verhalten zu erfassen.

Im Anwendungsfall der Malware-Forensik sollte die Analyse idealerweise innerhalb einer physischen Umgebung mit nativer Betriebssysteminstallation durchgeführt werden, wenn bekannt ist, dass die zu untersuchende Malware die Ausführung innerhalb einer virtuellen Umgebung oder Sandbox detektiert. Ein vorhandener Internetzugriff sollte entsprechend eingeschränkt und kontrolliert werden, um das

Schadensrisiko an Eigen- und Fremdsystemen zu minimieren. Im Idealfall wird der Internetzugriff schrittweise gewährt.

Zum Informationsaustausch der Analysedaten und -ergebnisse sollten definierte Schnittstellen festgelegt und verwendet werden. Nach Abschluss der Analysen, ist die Analyseumgebung vollständig zurückzusetzen, um nachfolgende Analysen nicht zu beeinträchtigen.

5.2 Methodik der Aufzeichnung

Bei der Auswahl einer geeigneten Methodik zur Aufzeichnung der Events sind Rahmenbedingungen zu berücksichtigen. Im Umgang mit den Sitzungen und Providern gelten die genannten Einschränkungen in Kapitel 3.9.2, wie beispielsweise die zulässige Anzahl gleichzeitig ausgeführter Sitzungen und maximaler Zuordnung von Providern zu einzelnen Sitzungen. Für die Aufzeichnung von ETW-Events müssen die ausgewählten und für die Analyse einzubeziehenden Provider einer oder mehreren ETW-Sitzungen zugeordnet und aktiviert werden.

Klassischerweise handelt es sich bei den gelieferten Events aus ETW zunächst um temporär zugängliche Daten, die zugleich flüchtig sind. Durch Wegschreiben und Auslagern der Sitzungspuffer in Event Trace Log (ETL) - Dateien lassen sich die Events auch persistent halten. In diesem Fall gilt es die, in der Sitzungskonfiguration festgelegten maximalen Dateigrößen zu berücksichtigen, um ein Überschreiben bereits erfasster Events zu verhindern [PB19].

Im Rahmen einer Live-Forensik sollen zur Laufzeit eines Prozesses die forensisch relevanten Daten und Informationen erhoben und gesammelt werden, die sich im späteren Verlauf zur Bestimmung des Prozessverhaltens eignen lassen. Eine Post-Mortem Analyse von Prozessverhalten mittels ETW schließt sich auf Grund der in Kapitel 3 beschriebenen Eigenschaften von Prozessen und der Einschränkungen von ETW zunächst kategorisch aus. Liegen die mittels Live-Response erhobenen Informationen über den Prozess aus ETW vor, lässt sich eine Post-Mortem Analyse im Nachhinein jedoch durchführen.

Bei der Aufzeichnung der Events aus den Providern kann in einer Sitzung festgelegt werden, ob eine Echtzeitaufzeichnung oder die Umlenkung der Events in eine Protokolldatei (ETL-Datei) erfolgen soll [Mic18a]:

- **Echtzeitaufzeichnung** (engl. *real time*): Die Events aus den Providern werden in Echtzeit analysiert und relevante Events aus dem Datenstrom extrahiert. Diese Variante ist mit der Möglichkeit zur Vorfilterung gegenüber der Umlenkung in eine ETL-Datei ressourcenschonender. Jedoch muss hierbei sichergestellt sein, dass alle Events zur Laufzeit rechtzeitig geparkt und verarbeitet werden, damit keine Events verloren gehen [Mic18a].
- **Protokolldatei** (engl. *file based*): Der Datenstrom der Events aus den Providern wird in eine ETL-Datei geschrieben. Die binärcodierten Daten können nach Abschluss der Aufzeichnung geparkt werden. Gegenüber der Methodik zur Echtzeitaufzeichnung bietet diese den Vorteil, dass sämtliche Informationen auch nach der Analyse noch im Rohdatenformat vorliegen und bei Bedarf zusätzliche Daten extrahiert werden könnten [Mic20e].

Für die Aufzeichnung des Prozessverhaltens sollte die Echtzeitaufzeichnung einer Umlenkung der Events in eine ETL-Datei vorgezogen werden, da die Analysesitzung die aus der Sitzung gestreamten Events direkt weiterverarbeiten kann und somit einem Verlust von Events vorgebeugt wird. Der Zugriff auf den Arbeitsspeicher ist aus Sicht des Analyseprogrammes performanter als der Zugriff auf den Sekundärspeicher (z.B. Festplatte). Hierdurch kann den negativen Eigenschaften von ETW entgegengewirkt werden. Bei ETL-Dateien handelt es sich um binärcodierte und komprimierte Logdateien, deren Größe mit Anzahl der aufgezeichneten Events steigt [Mic18a; Mic20e; Mic16b].

Wie in Kapitel 5 beschrieben, liegt der Fokus in der Beobachtung und Auswertung des Verhaltens eines bestimmten zu untersuchenden Prozesses. Daher sollten sich die aufgezeichneten Events auf diesen Untersuchungsgegenstand begrenzen. Um die Ressourcen des Analyseystems nicht durch die Aufzeichnung von nicht in Relation stehender Events mit dem Prozess zu belasten, werden Filterregeln bei der Aufzeichnung eingerichtet. Die Filterregeln können an verschiedenen Positionen platziert werden. Bereits zur Aufzeichnung der Events können diverse Filter an verschiedenen Stellen positioniert werden, um die zu erhebenden Daten auf die relevanten Informationen zu begrenzen. Aus den einzelnen ETW-Providern werden standardmäßig Events aller aus dem System ausgeführter Prozesse geliefert. Durch die Angabe von Keywords können gezielt bestimmte Event-Gruppen aus den Providern herausgefiltert werden. Die Keywords sollten sich an dem zu beobachteten Verhalten orientieren. Im Standardfall wird keine Filterung der Keywords vorgenommen.

Nachdem die Events prozessbezogen aus dem Datenstrom der Sitzung extrahiert wurden, sollen die in der Analyse als relevant definierten Ereignisse in eine CSV-Datei geschrieben werden.

Erst nach gestarteter Sitzung werden die Events aus den Providern an die Sitzung weitergereicht und aufgezeichnet. Alternativ oder zusätzlich lassen sich bestehende und ausgeführte Sitzungen zur Laufzeit in die Analyse miteinbeziehen. Dies trifft beispielsweise auf die NT Kernel Logger Sitzung zu, welche als zirkulierende Sitzung verschiedene Events aus dem Kernel erfasst. Die Events werden dabei in einem Cache zwischengespeichert und zyklisch [Sou12].

Wegen der Vielzahl möglicher Events und Providern ist die Bestimmung der Filterposition entscheidend. Auf Providerebene ist eine Vorfilterung der zu erfassenden Events ausschließlich mittels der festgelegten Keywords und (Log-)Leveln möglich [Gra18]. Erst durch das Eventparsing lassen sich weitere Filterregeln anwenden. Anstelle des Wegschreibens aller erfassten Events kann durch eine gezielte Filterung der Fokus auf die Events gelegt werden, die sich zum einen auf den zu beobachtenden Prozess beschränken und zum anderen auf die vermuteten Verhaltensweisen ausrichten. Im Bedarfsfall könnten die Filter dynamisch erweitert werden.

Der Zeitraum der Aufzeichnung sollte vorab anhand unterschiedlicher Kriterien festgelegt werden:

- (a) Der untersuchte Prozess ist terminiert.
- (b) Der untersuchte Prozess hat seine Aktivität eingestellt.
- (c) Das Ende der festgelegten Untersuchungszeitspanne ist erreicht.

Ferner sind die Kriterien auch auf die möglicherweise erstellten Kindprozesse des beobachteten Prozesses zu übertragen, um im Falle zusätzlicher Prozesserzeugung ein ganzheitliches Verhalten zu erfassen.

5.3 Identifikation und Auswahl geeigneter ETW-Provider

ETW-Provider liefern als zentrale Datenquelle die ETW-Events, die relevante Informationen und Rückschlüsse über das Verhalten von Prozessen geben können. Wie eingehend in Kapitel 4.4.1 erwähnt, sind werksseitig in einem Windows 10 Betriebssystem über 1000 registrierte ETW-Provider implementiert. Die Anzahl der tatsächlich vorhandenen und registrierten Provider in einem Windows 10 Betriebssystem ist allerdings nicht statisch und kann variieren, wenn über den Programmcode von Anwendungen, z.B. durch Verwendung der Win32-API Funktion `EventRegister`,

zusätzliche (anwendungsspezifische) Provider registriert wurden [Sou12, S. 448]. Zu unterscheiden ist zwischen Providern, die durch das Betriebssystem selbst (sog. System ETW-Provider) und durch (Drittanbieter-)Anwendungen bereitgestellt werden (sog. User bzw. Custom ETW-Provider). System ETW-Provider sind überwiegend in Bibliotheken und Komponenten des Betriebssystems verankert, sodass (System-)Funktionsaufrufe und andere Aktivitäten eines ausgeführten Prozesses in Relation mit Events aus den System ETW-Provider gesetzt werden können. Eine (Drittanbieter-)Software muss daher zwangsläufig keine eigenen benutzerdefinierten ETW-Provider registrieren, damit prozessbezogene Events über die zugeordneten Prozesse geliefert werden. [Mic16a]

In Hinblick auf Prozesse mit einer schadhafte Absicht (z.B. einer Malware), kann in den meisten Fällen davon ausgegangen werden, dass ihre Entwickler keine eigenen Provider im System registrieren oder eigenständig Events in andere Provider schreiben. Eine Verhaltensanalyse sollte sich daher zunächst auf die Auswertung prozessbezogener Events aus den registrierten System ETW-Providern konzentrieren. Zu diesen wurden bereits in vorausgehenden Forschungsarbeiten (siehe Kapitel 4.1.5) relevante Provider als Datenquelle zur Unterstützung forensischer Analysen identifiziert. Einige der Provider werden zudem bereits für Host-Intrusion Detektion Systeme über das Event Logging verwendet [nxl20]. Für diese sollte daher überprüft werden, inwiefern sie sich für die gezielte Verhaltensanalyse von Prozessen hinzuziehen und einsetzen lassen. Außerdem sollten weitere zur Verfügung stehende Provider mit in die Analysen einbezogen werden, sofern zu erwarten ist, dass ihre (prozessbezogenen) Events relevante Erkenntnisse über das Prozessverhalten liefern.

Der Zugriff auf Events aus den ETW-Providern ist an Berechtigungen geknüpft. Standardmäßig können nur Administratoren, Benutzer der Log-User Gruppe sowie Dienste im Kontext LocalSystem, LocalService, NetworkService ETW-Sitzungen verwalten und Events konsumieren. Die NT Kernel Logger Sitzung kann ausschließlich mit administrativen Berechtigungen oder als LocalSystem ausgeführtem Dienst gesteuert werden [Mic18g].

Eine Vielzahl der registrierten ETW-Provider verfügt neben der GUID (siehe Kapitel 3.9.2) über einen Bezeichner (Providername), der bereits erste Hinweise auf den möglichen Inhalt der zu erwartenden Events geben kann. ETW-Provider ohne Bezeichner werden lediglich durch ihre GUID repräsentiert. Häufig handelt es sich bei solchen Providern um WPP Software Tracing Provider oder des sog. TraceLoggings. Diese Provider unterscheiden sich insofern von denen der registrierten Provider, da das zugehörige Manifest der Events nicht systemseitig abrufbar ist, sondern aus

TMF oder PDB-Symbolen extrahiert werden muss (siehe Kapitel 4.4.1). Bei nicht-manifest basierten Providern können lediglich Keywords ermittelt werden, nicht jedoch Informationen über die Datenstrukturen der Events. [Gra18]

5.4 Aufzeichnen des Prozessverhaltens

5.4.1 Anlegen der ETW-Analysesitzung

Für die Aufzeichnung von Events ist zunächst erforderlich, die für die Analyse ausgewählten ETW-Provider durch Zuordnung zu einer neuen ETW-Sitzung (Analysesitzung) zu aktivieren. Zur Aufzeichnung der Analyse wird mindestens eine neue ETW-Sitzung im Analysesystem angelegt. Im Bedarfsfall kann es sich hierbei auch um mehrere ETW-Sitzungen handeln. Bestehende ETW-Sitzungen, wie die NT Kernel Loggers müssen und können technisch nicht neu angelegt werden. Die Events aus den zugeordneten Providern dieser Sitzungen können bei Bedarf in die Analyse miteinbezogen werden. Zur Beobachtung des Prozessverhaltens sollen die Events aus der NT Kernel Logger Sitzung berücksichtigt werden, aus der wichtige Erkenntnisse aus Kernelsicht gewonnen werden können (Kapitel 3.9.2). [Mic18a; Mic20e]

5.4.2 Zuordnen von ETW-Providern zur Analysesitzung

Die für die Analyse ausgewählten ETW-Provider werden der ETW-Analysesitzung zugeordnet. Hierbei können nachfolgende Informationen berücksichtigt werden und eine erste Filterung erfolgen [Gra18]:

- GUID oder Bezeichner des ETW-Providers
- Level [*optional zur Vorfilterung*]
- Keywords [*optional zur Vorfilterung*]

Als weitere Option ist es möglich die, durch den Provider, zu liefernden Events hinsichtlich bestimmter Eigenschaften, wie Prozess-ID oder Prozessname zu filtern. Von diesen Optionen sollte im Fall der Prozessverhaltensanalyse Gebrauch gemacht werden, um die zu erhebende und zu verarbeitende Datenmenge zu reduzieren.

5.4.3 Konfiguration der Aufzeichnungsmethodik

Es wird vorab festgelegt, ob die Events in Echtzeit ausgewertet werden oder eine Event Trace Logdatei (ETL) erstellt werden soll, aus der sich (weitere) Events

zu späterem Zeitpunkt extrahieren lassen. Mittels speziellen Autologger-Sitzungen können zusätzlich Ereignisse während des Boot- oder Shutdownvorgangs des Betriebssystems festgehalten werden [Mic18f; Sou12, S. 449]. Dies ist für Prozesse relevant, deren Aktivitäten innerhalb solcher Zeiträume beobachtet werden sollen. Die Aufzeichnungsmethodik wird gemäß der gewählten Strategie (siehe Kapitel 5.2) konfiguriert. Für die Analyse von Prozessverhalten wird eine Analysesitzung erstellt, die eine Echtzeitaufzeichnung der Events aus den später zugeordneten ETW-Providern durchführt. Insbesondere die bereits aus anderen Forschungsarbeiten identifizierten und bekannten forensisch relevanten Provider sowie zusätzlich weitere Provider, die mit dem Prozess in Relation stehen, sollen in der Analysesitzung miteinbezogen werden. Neben den ETW System Providern, wird außerdem die Kernelsitzung mit allen Keywords über die Analysesitzung aufgezeichnet.

5.4.4 Starten der Aufzeichnung durch Aktivierung der Analysesitzung

Die Analysesitzung wird für die Aufzeichnung des Prozessverhaltens gestartet. Hierbei ist zu berücksichtigen, dass administrative Berechtigungen erforderlich sind. Da die Analysesitzung prozessbezogene Ereignisse verarbeiten soll, muss die Prozess-ID oder der Prozessname des zu überwachenden Prozesses angegeben werden.

5.4.5 Prozess starten und überwachen

Der zu beobachtende Prozess wird gestartet. Die Zuweisung der Prozess-ID durch das Betriebssystem sollte frühzeitig protokolliert werden, um sie als Parameter für die Filterung verwenden zu können. Dies lässt sich automatisiert durch die Beobachtung der zugehörigen Events zur Prozesserstellung aus der Kernelsitzung erfassen. Alternativ lässt sich auch ein bereits ausgeführter Prozess beobachten. Hierzu wird die Prozess-ID zur Laufzeit ermittelt und für die Filterung der Events verwendet. Events werden in diesem Fall erst zum Aufzeichnungszeitpunkt erfasst, sodass keine Beobachtung des Verhaltens über die gesamte Prozesslaufzeit vorgenommen wird.

5.4.6 Aufzeichnung beobachten und Events filtern

Während der Aufzeichnung der ETW-Events aus den Providern können vorab festgelegte Verhaltensweisen bereits als Zwischenergebnisse über eine Konsolenausgabe sichtbar gemacht werden. Situationsabhängig lässt sich eine Aufzeichnung dynamisch zur Laufzeit anpassen. Beispielsweise kann das Erzeugen von Kindprozessen

durch den beobachteten Prozess eine Ausweitung der Filterregelung auf weitere Prozess-IDs auslösen. Zur Laufzeit der Analyse wird bereits eine Filterung der aufgezeichneten Events aus der Sitzung auf prozessbezogene Events vorgenommen. Diese herausgefilterten Ereignisse werden nach Abschluss der Analyse in eine CSV-Datei exportiert.

5.4.7 Stoppen der Aufzeichnung nach Terminierung des Prozesses

Nach den vorab festgelegten Kriterien wird die Aufzeichnung der Events manuell oder automatisch gestoppt, in dem die ETW-Analysesitzungen beendet werden. ETW-Sitzungen überleben die Terminierung des Prozesses, durch den diese initiiert wurde. Im Falle der Echtzeitaufzeichnung nimmt ein Fortbestehen der Sitzung weiterhin Systemressourcen ein [Mic19b; Mic20e]. Für die Aufzeichnung in eine ETL-Datei bedeutet dies, dass weiterhin eine hohe Anzahl von Schreib-/Leseoperationen auf das Speichermedium erfolgen und bei falscher Konfiguration das Speichermedium vollgeschrieben wird. Verwaiste Sitzungen sollten daher durch das automatische Beenden der Sitzungen nach Terminierung der Analysesitzung vermieden werden.

Nach Aufzeichnungsende verbleiben alle erfassten Events aus der Analysesitzung zugeordneten Provider je nach Sitzungskonfiguration in der ETL-Datei (Protokolldateiaufzeichnung) oder die während der Aufzeichnung bereits extrahierten prozessbezogenen Ereignisse aus dem Datenstrom in einer Logdatei, z.B. im CSV-Format. Das CSV-Format bietet sich gegenüber einem Log auf Textdateibasis aufgrund einer vereinheitlichten Strukturierung einzelner Werte und Datenfelder an. Hierdurch lassen sich die Daten weiterverarbeiten oder durch zusätzliche Werkzeuge filtern.

5.5 Auswertung der Aufzeichnung

Die aufgezeichneten und prozessbezogenen ETW-Events werden als Export der Analysesitzung ausgewertet und hinsichtlich der Verhaltensweisen untersucht. Anhaltspunkte bieten hier vor allem Events aus den bereits als relevant identifizierten ETW-Providern. Es sollte besonderes Augenmerk auf das vorab vermutete Verhalten gelegt werden, um später bewerten zu können, aus welcher Absicht ein bestimmtes Verhalten durch den zu beobachteten Prozess erfolgte. Für die Auswertung sollten die erhobenen Informationen aus den Events der einzelnen Provider mittels Filterregeln, z.B. durch ein Tabellenkalkulationsprogramm oder alternative Werkzeuge durchsucht werden. Bei der Auswertung bedürfen festgestellte (Prozess-)Übergänge

einer weiteren (Nach-)Untersuchung. Insbesondere trifft dies zu, wenn durch den beobachteten Prozess weitere Prozesse erzeugt wurden oder über die verschiedenen Schnittstellen ein Übergang feststellbar ist.

Markante Hinweise auf das Prozessverhalten werden entsprechend als relevante Information gekennzeichnet und können bereits in eine zeitliche Korrelation zu anderen Verhaltensweisen gebracht werden. Gleichzeitig können hierbei die forensisch relevanten Provider identifiziert und zugeordnet werden.

5.6 Bewertung der Ergebnisse

Auf Basis der ausgewerteten Aufzeichnung zum beobachteten Prozessverhalten lässt sich mit Hilfe der gekennzeichneten Datensätze der exportierten CSV-Datei beurteilen, welche der erfassten Verhaltensweisen und Aktivitäten einer bestimmten Absicht zugrunde liegen. Diese lässt sich gezielt nach bestimmten Providern oder Inhalten der Eventinhalte filtern. Mit Hilfe der CSV-Datei können die Daten beliebig weiterverarbeitet werden. Durch Bestimmung der Ziele und Absichten der Verhaltensweisen, lässt sich legitimes Verhalten von einer unerwünschten oder schadhafte Absicht abgrenzen. Die gewonnen Erkenntnisse werden bewertet und dokumentiert. Für spätere Bewertungen stellen die identifizierten Provider einen Anhaltspunkt dar, ob diese auch zukünftig miteinbezogen werden sollten. Gleichzeitig können Empfehlungen für zukünftige Filterregeln und -optionen getroffen werden.

6 Proof-of-Concept (PoC): Prozessanalyse mit Hilfe von ETW

6.1 Exemplarischer Anwendungsfall: DemoProcess (PoC I)

Zur Demonstration der prinzipiellen Vorgehensweise und Durchführbarkeit einer Prozessanalyse mit Hilfe von ETW wird ein Beispielprozess („DemoProcess“) beschrieben und entwickelt, durch den verschiedene Verhaltensweisen testweise abgedeckt werden. Gleichzeitig soll anhand dieses exemplarischen Anwendungsfalls eine erste Evaluation der beschriebenen Vorgehensweise in Kapitel 5 vorgenommen werden. Bei der Entwicklung des Anwendungsfalls und Konzeption der Testfälle soll sich repräsentativ am Verhalten orientiert werden, welches sich zugleich auch in gängiger Malware beobachten lässt. Eine Eigenentwicklung des Beispielprozesses bietet den Vorteil der Quellcodehoheit und somit einer individuell anpassbaren Gestaltung des Funktions- und Verhaltensumfangs. Kenntnis über den Quellcode ermöglicht darüber hinaus einen gezielten Vergleich der zu erwartenden Ergebnisse aus dem PoC, um später forensisch relevante ETW-Provider ermitteln zu können. In einem zweiten Schritt kann die Konfiguration des PoC I auch auf einen konkreten Anwendungsfall, wie einer Malware (siehe PoC II und III), übertragen und evaluiert werden.

Der exemplarische Anwendungsfall wird in der Programmiersprache C++ als x64-Konsolenanwendung und vorzugsweiser Nutzung der Win32-API Bibliotheken (siehe 3.6.1) programmiertechnisch umgesetzt. Die festgelegten Testverhaltensweisen TV1 bis TV4 sollen nach Prozessstart durch Benutzereingabe automatisiert und aufeinanderfolgend zeitversetzt durchlaufen werden, um die Testdaten und -events für das PoC zu produzieren. Für die Nutzung der Win32-API wird auf die von Microsoft bereitgestellten Beispiele zur Nutzung der Bibliotheken zurückgegriffen und sofern erforderlich (geringfügige) Anpassungen vorgenommen.

Nachfolgend werden die einzelnen Testverhaltensweisen kurz anhand von vereinfachten Quelltextauszügen vorgestellt. Der vollständige Quellcode über den Beispielprozess („DemoProcess“) ist Anhang A.2 zu entnehmen.

TV1: Abfragen und Setzen von Registrierungsschlüsseln und -werten

(Malware-)Prozesse stellen häufig Abfragen an die Registrierungsdatenbank, um Informationen über die Betriebssystemkonfiguration und der Plattform zu erhalten, unter der sie ausgeführt werden. Insbesondere Malware versucht zu erkennen, ob die Ausführung innerhalb einer virtuellen Umgebung oder einer Sandbox erfolgt, um entsprechende Analysen frühzeitig zu detektieren und ihr Verhalten anzupassen oder eine weitere Ausführung zu unterbrechen.

In der Taktik „Defense Evasion“ der MITRE ATT&CK wird mit der Technik Virtualization/Sandbox Evasion (ID T1497) exemplarisch beschrieben, wie Angreifer eine virtuelle Umgebung oder Sandbox erkennen können, um sich vor einer potentiellen Analyse zu schützen, welche die Verhaltensweisen und Tools offenlegen könnten. Anhaltspunkte und Artefakte lassen sich aus der Registrierung oder dem Dateisystem entnehmen [MIT20e].

Beispielsweise sollen im TV1 daher folgende Registrierungsschlüssel und Werte abgefragt werden:

- HKLM\SOFTWARE\Oracle\VirtualBox Guest Additions
- HKLM\HARDWARE\Description\System\SystemBiosVersion;"VMWARE"
- HKLM\HARDWARE\ACPI\DSDT\BOX_

Ein Auszug der programmiertechnischen Umsetzung von TV1 im exemplarischen Anwendungsfall (DemoProcess) kann dem Listing 6.1 entnommen werden:

```

1 RegOpenKeyEx(HKEY_LOCAL_MACHINE, L"SOFTWARE\\Oracle\\VirtualBox Guest Additions", 0, KEY_READ, &
    phkResult);
2 RegCloseKey(phkResult);
3 [...]
4 RegOpenKeyEx(HKEY_LOCAL_MACHINE, L"HARDWARE\\Description\\System", 0, KEY_READ, &phkResult);
5 RegQueryValueExW(phkResult, L"SystemBiosVersion", 0, NULL, LPBYTE (value), &BufferSize);
6 RegCloseKey(phkResult);
7 [...]
8 RegOpenKeyEx(HKEY_LOCAL_MACHINE, L"HARDWARE\\ACPI\\DSDT\\BOX_", 0, KEY_READ, &phkResult);
9 RegCloseKey(phkResult);

```

Listing 6.1: DemoProcess - TV1 Abfragen von Registrierungsschlüsseln und -werten

Um sich eine Persistenz im Opfersystem zu verschaffen, wird durch einen Malwareprozess der Versuch unternommen, die sog. Run Keys bzw. Startup Folders in der Registry zu manipulieren und dort ein Programm zu hinterlegen, das nach Benutzeranmeldung automatisch gestartet und ausgeführt wird (Autostart). Diese Technik wird in ID1060 der ATT&CK Matrix beschrieben und empfiehlt zur Detektion ein Monitoring der zugehörigen Registry-Pfade [MIT20a]. Im Pfad HKEY_CURRENT_USER\Software

\Microsoft\Windows\CurrentVersion\Run soll daher im TV1 ein Eintrag vorgenommen werden.

Ein vereinfachter Auszug der programmiertechnischen Umsetzung des Setzens von Registrierungsschlüsselwerten von TV1 im exemplarischen Anwendungsfall (DemoProcess) kann dem Listing 6.2 entnommen werden:

```
1 RegOpenKeyEx(HKEY_CURRENT_USER,L"SOFTWARE\\Microsoft\\Windows\\ CurrentVersion\\Run", 0, KEY_READ, &
    phkResult);
2 RegSetValueExW(phkResult,L"C:\\TV1DemoRunKey.exe",0, REG_SZ, (LPBYTE) data, wcslen(L"TV1DemoRunKey")
    * sizeof(TCHAR));
3 RegCloseKey(phkResult);
```

Listing 6.2: DemoProcess - TV1 Setzen von Registrierungsschlüsseln

TV2: Nachladen einer Datei von einem Webserver

Um Dateien, wie ausführbare Dateien oder Schadcode nachzuladen, wird in TV2 die Win32-API Funktion `URLDownloadToFile` verwendet. Die Datei `favicon_hsw_fiw.png` wird vom Server `fiw.hs-wismar.de` heruntergeladen und als `favicon.png` im aktuellen Ausführungspfad des Beispielprozess gespeichert. Die Technik „Remote File Copy“ wird als Command and Control und Lateral Movement Taktik unter der ID T1105 der MITRE ATT&CK eingeordnet [MIT20d].

Auszug der Umsetzung im exemplarischen Anwendungsfall von TV2, in dem das Favicon der Hochschule Wismar heruntergeladen wird (Listing 6.3):

```
1 URLDownloadToFile(NULL, L"https://fiw.hs-wismar.de/storages/hs-wismar/favicons/favicon_hsw_fiw.png",
    "favicon.png",0,NULL);
```

Listing 6.3: DemoProcess - TV2 Nachladen einer Datei von einem Webserver

TV3: Lesen- und Schreiben einer (Text-)Datei

Repräsentativ wird eine Textdatei (Dateiname: „`TV3CreateFileTestFile.txt`“) im Ausführungspfad des Beispielprozesses erstellt (`CreateFile()`) und mit dem Inhalt „`TV3 TestText1234`“ beschrieben (`WriteFile()`). Der Dateiinhalt wird anschließend ausgelesen (`ReadFile()`) und auf der Konsole ausgegeben.

Malwareprozesse könnten die Funktionen beispielsweise auch zur Obfusking einsetzen, indem ein verschlüsselter Inhalt aus den heruntergeladenen Daten (siehe TV2), entschlüsselt in eine neue Datei geschrieben wird (siehe ID T1022 Data Encrypted der MITRE ATT&CK Matrix [MIT18]).

Auszugsweise wird in Listing 6.4 der vereinfachte Quellcode zur programmiertechnischen Umsetzung von TV3 im exemplarischen Anwendungsfall aufgeführt:

```

1 HANDLE hfile = CreateFile(L"TV3CreateFileTestFile.txt", GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ, NULL, CREATE_NEW, FILE_ATTRIBUTE_NORMAL, NULL);
2 [...]
3 WriteFile(hFile, DataBuffer, dwBytesToWrite, &dwBytesWritten, NULL);
4 [...]
5 ReadFileEx(hFile, ReadBuffer, sizeof(ReadBuffer)-1, &ol, NULL);
6 [...]
7 CloseHandle(hFile);

```

Listing 6.4: DemoProcess - TV3 Lesen- und Schreiben einer (Text-)Datei

TV4: Erstellung eines Kindprozesses

Es wird ein neuer (Kind-)Prozess mittels des Win32-API Funktionsaufrufes `Create Process` gestartet und ausgeführt. In der Praxis werden bereits auf dem Dateisystem vorhandene ausführbare Dateien, wie beispielsweise von Komponenten des Betriebssystems (Windows Executables) ausgeführt. Angreifer bedienen sich der Technik T1055 „Process Injection“, um anschließend fremden Programmcode im Adressbereich und Kontext eines anderen (legitimen) Prozesses auszuführen [MIT20c]. Im Beispielprozess wird lediglich das Anzeigeprogramm zur verwendeten Windows Version (`winver.exe`) gestartet.

Listing 6.5 stellt einen vereinfachten Auszug aus der programmiertechnischen Umsetzung von TV4 im exemplarischen Anwendungsfall dar:

```

1 CreateProcess( L"C:\\Windows\\System32\\winver.exe", NULL, NULL, NULL, 0, NULL, NULL, &sInfo , &
    pInfo);

```

Listing 6.5: DemoProcess - TV4 Erstellung eines Kindprozesses

6.1.1 Vorbereitung und Fokus der Analyse

Im Rahmen des Proof-of-Concepts (PoC) wird in Vorbereitung auf die durchzuführende Analyse mit Hilfe eines im Rahmen dieser Arbeit zu entwickelnden Demonstrator (Analysewerkzeug), der exemplarische Anwendungsfall untersucht. Durch Kenntnis und Wissen über die verwendeten Funktionen und Verhaltensweisen des selbst entwickelten Beispielprozesses, liegt der Fokus auf der Beobachtung und Erkennung der vorab festgelegten Verhaltensweisen (TV1-TV4), die sich auf Prozessoperationen in der Windows-Registrierungsdatenbank, dem Dateisystem, dem Netzwerk und

Prozesserstellung erstrecken. Ziel ist es, geeignete ETW-Provider und Events zu identifizieren, welche den Verhaltensweisen TV1-TV4 eindeutig zugeordnet werden.

Das kompilierte Programm des exemplarischen Anwendungsfalls liegt als Portable Executable (PE)-Datei („`DemoProcess.exe`“) vor (Screenshot: Anhang B.2) und lässt sich für die Analyse kontrolliert auf einem Analysesystem zur Ausführung bringen. Die Analyse soll über die gesamte Laufzeit des Prozesses durchgeführt werden. Die zu erhebende Datenmenge beschränkt sich auf die dem Prozess zurechenbaren Ereignisse. Hierzu sollen die Events nach Prozess-ID gefiltert werden.

Zur Durchführung der Prozessanalyse des PoCs wird das Analysesystem innerhalb einer virtuellen Instanz mit Windows 10 x64 Education (Version 1909, OS Build 10.0.18363.836) unter der Virtualisierungsplattform Oracle VirtualBox eingerichtet. Die verwendeten Konfigurationsparameter der virtuellen Maschine können Anhang E.1 entnommen werden. Das .NET Framework 4.5 wird zur Ausführung des Analysewerkzeugs benötigt und ist standardmäßig bereits vorinstalliert. Der Windows Defender wird deaktiviert. Die spezielle Betriebssystemkonfiguration wird im Anhang E.2 aufgelistet. Der Internetzugriff und Netzwerkverkehr wird nicht restriktiert oder gesondert mitgeschnitten.

Um hierbei insbesondere die beteiligten ETW-Provider, welche Events zum Beispielprozess liefern, zu ermitteln, sollen für diese Analyse nach dem Maximalprinzip alle im System-registrierten ETW-Provider miteinbezogen werden. Die Aufzeichnung der Events aus den ETW-Providern soll mit Hilfe des im Rahmen der vorliegenden Thesis entwickelnden Analysewerkzeugs (ETWProcessTracer, siehe Kap. 4.3.3) automatisiert durchgeführt werden. Bei Aufruf des ETWProcessTracers ist als Konsolenparameter `--PID` mindestens die Prozess-ID des Prozesses anzugeben, der den Ausgangspunkt der Analyse darstellen soll. Um alle Provider nach dem Maximalprinzip in die Analyse einzubeziehen, wird für die Kernel Provider der Parameter `--KernelKeywords All` angegeben und der Parameter `--ProviderConfig` für die im System registrierten Provider nicht abweichend konfiguriert. Über optionale Parameter lassen sich weitere Optionen, wie ein Zeitlimit für den Timer oder einen Pfad zur individuell festgelegten Liste der einzubeziehenden ETW-Provider angeben. Ein Screenshot des ETWProcessTracers ist im Anhang B.3 abgebildet.

Neben dem Analysewerkzeug wird zusätzlich ein weiteres Werkzeug verwendet, das den Beispielprozess zunächst als suspendierten Prozess startet. Der Hintergrund für diese Form des Prozessstarts über den `ProcessStartHelper` (Quellcode siehe Anhang A.3) besteht darin, dass eine PID durch das Betriebssystem erst im Rahmen

der Prozesserstellung vergeben wird und erst im Anschluss an das Analysewerkzeug übergeben werden kann. In dieser Zeitspanne werden zum entsprechenden Prozess bereits Events ausgeliefert, die in der Analyse nicht mit aufgezeichnet werden könnten. Dem `ProcessStartHelper`, als konsolenbasierte Anwendung, wird der Pfad zur ausführbaren Datei übergeben, die als Prozess ausgeführt werden soll (Listing 6.6):

```
C:\> ProcessStartHelper.exe DemoProcess.exe
```

Listing 6.6: Ausführung des DemoProcess mit dem ProcessStartHelper

6.1.2 Methodik der Aufzeichnung

Während der Laufzeit des Beispielprozesses wird durch den `ETWProcessTracer` eine Echtzeitaufzeichnung der ETW-Analysesitzung durchgeführt. Alle prozessbezogenen Events innerhalb der Analysesitzung werden aus dem Datenstrom herausgefiltert, um sie in nach dem Parsen der Protokolldatei (CSV-Format) zuzuführen. Die in Kapitel 4.4 aufgeführten Herausforderungen im Umgang mit ETW werden berücksichtigt. Insgesamt wird eine ETW-Sitzung benötigt, die als Analysesitzung fungiert. Dieser werden alle registrierten ETW-Provider im Analysesystem sowie der verfügbaren Kernel-Provider zugeordnet. Nicht aktivierbare ETW-Provider werden zur späteren Nachvollziehbarkeit protokolliert. Die Aufzeichnung wird bis zur Terminierung des Beispielprozess und der erzeugten Kindprozesse durchgeführt.

6.1.3 Identifikation und Auswahl geeigneter ETW-Provider

Die auf dem Analysesystem registrierten und somit für die Analyse zur Verfügung stehenden ETW-Provider lassen sich mit Hilfe des in Windows 10 integrierten Werkzeugs `logman` (siehe Kapitel 4.2.2) auflisten (Listing 6.7):

```
C:\> logman query providers
```

Provider	GUID
ACPI Driver Trace Provider	{DAB01D4D-2D48-477D-B1C3-DAAD0CE6F06B}
[...]	
Microsoft-Windows-DNS-Client	{1C95126E-7EEA-49A9-A3FE-A378B03DDB4D}
Microsoft-Windows-Documents	{C89B991E-3B48-49B2-80D3-AC000DFC9749}
[...]	
XWizard Framework	{777BA8FF-2498-4875-933A-3067DE883070}

Listing 6.7: Auszug aus der Auflistung aller im System registrierter ETW-Provider

Die Konsolenausgabe liefert eine Übersicht aller im System registrierten Provider mit Namen und ihrer zugeordneten GUID. Die Benennung der ETW-Provider folgt keinem einheitlich vorgegebenen Namensschema. Programmier technisch lassen sich die im System registrierten ETW-Provider alternativ über den Aufruf von `TraceEventProviders.GetPublishedProviders()` der Trace Event Bibliothek als Liste zurückgeben. Auf dem Analysesystem stimmten die erfassten registrierten Provider zwischen den beiden Abfragevarianten überein.

Mit Hilfe des Parameters `query providers "ProviderName"` des Werkzeugs `logman` (Listing 6.8) können zusätzlich die verfügbaren Keywords und Level mit zugehöriger Beschreibung abgefragt werden. Diese bieten erste Anhaltspunkte auf mögliche Events, die aus dem Provider geliefert werden können. Auszugsweise wird in Listing 6.8 eine Providerabfrage zum Provider Microsoft-Windows-DNS-Client dargestellt:

```
C:\>logman query providers "Microsoft-Windows-DNS-Client"
```

Provider	GUID
Microsoft-Windows-DNS-Client	{1C95126E-7EEA-49A9-A3FE-A378B03DDB4D}

Value	Keyword	Description
0x00000000000000100	ut:GenericEvent	
0x00000000100000000	ut:DnsAutoLogKeyword	
0x00000000200000000	ut:PolicyTable	
0x00000000400000000	ut:PerfCheckPoints	
0x00000000800000000	ut:RegistrationEvent	
0x00000001000000000	ut:SendPath	
0x00000002000000000	ut:ReceivePath	
[...]		

Value	Level	Description
0x02	win:Error	Error
0x03	win:Warning	Warning
0x04	win:Informational	Information

PID	Image
0x000016f0	C:\Windows\System32\backgroundTaskHost.exe
0x00002e5c	C:\Windows\SystemApps\Microsoft.Windows.Cortana_cw5n1h2txyewy\SearchUI.exe
0x000022c8	C:\Windows\System32\RuntimeBroker.exe
[...]	

Listing 6.8: Abfrage - Microsoft-Windows-DNS-Client Provider (Auszug)

Wie Listing 6.8 entnommen werden kann, sind zusätzlich auch die Prozesse aufgeführt, über die Events durch den Provider geliefert werden. Weiterführende Informationen zu einem Provider, wie den möglichen Events und ihren zugehörigen Datenstrukturen

lassen sich mit den Werkzeugen ETWExplorer (siehe Bild 6.1) und WEPEXplorer ermitteln (siehe Kapitel 4.3.2). Umgekehrt ist ebenfalls eine Liste zu einem Prozess zugeordneten Provider abrufbar (siehe Listing 6.9). Insbesondere die Strukturen der Events lassen sich für die Erstellung und Anpassung von Filterregeln heranziehen.

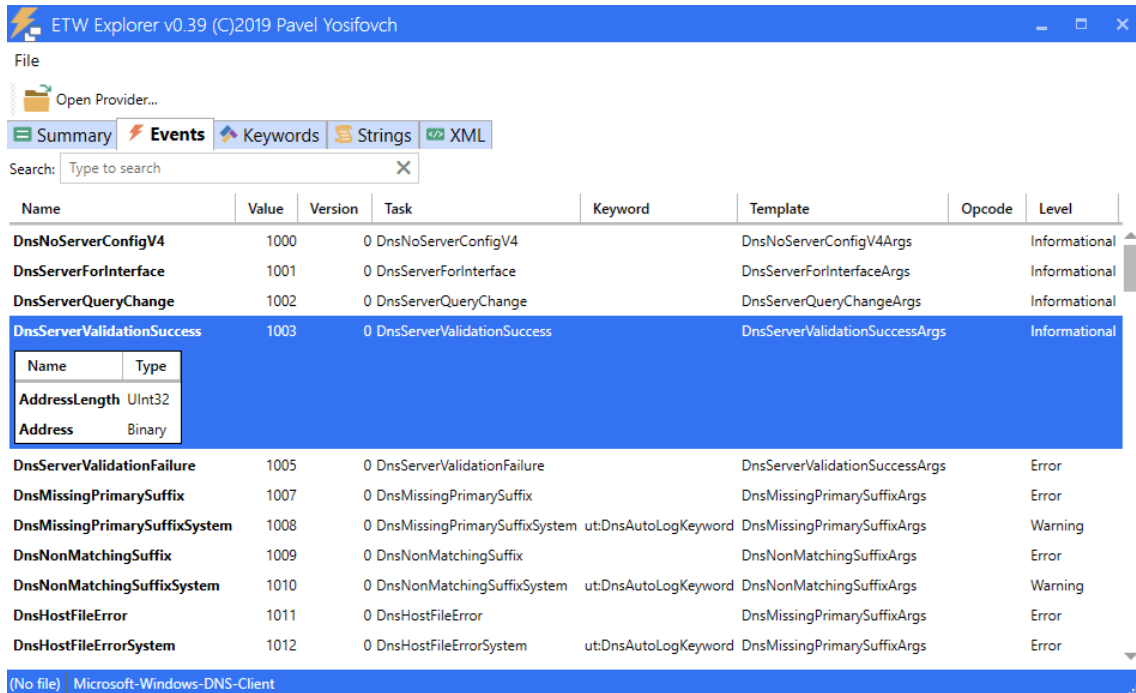


Bild 6.1: Events des Providers Microsoft-Windows-DNS-Client (ETWExplorer)

Für eine Übersicht der verknüpften ETW-Provider mit dem Prozess aus dem exemplarischen Anwendungsfall wird logman mit folgenden Parametern ausgeführt: logman query providers -pid [Prozess-ID].

```
C:\>logman query providers -pid 23868
```

Provider	GUID
FWPUCLNT Trace Provider	{5A1600D2-68E5-4DE7-BCF4-1C2D215FE0FE}
Microsoft-IEFRAME	{5C8BB950-959E-4309-8908-67961A1205D5}
Microsoft-Windows-AppModel-Runtime	{F1EF270A-0D32-4352-BA52-DBAB41E1D859}
Microsoft-Windows-ASN1	{D92EF8AC-99DD-4AB8-B91D-C6EBA85F3755}
Microsoft-Windows-AsynchronousCausality	{19A4C69A-28EB-4D4B-8D94-5F19055A1B5C}
Microsoft-Windows-CAPI2	{5BBCA4A8-B209-48DC-A8C7-B23D3E5216FB}
Microsoft-Windows-COM-Perf	{B8D6861B-D20F-4EEC-BBAE-87E0DD80602B}
Microsoft-Windows-COM-RundownInstrumentation	{2957313D-FCAA-5D4A-2F69-32CE5F0AC44E}
Microsoft-Windows-Crypto-BCrypt	{C7E089AC-BA2A-11E0-9AF7-68384824019B}
Microsoft-Windows-Crypto-NCrypt	{E8ED09DC-100C-45E2-9FC8-B53399EC1F70}
Microsoft-Windows-Crypto-RSAEnh	{152FDB2B-6E9D-4B60-B317-815D5F174C4A}
Microsoft-Windows-DNS-Client	{1C95126E-7EEA-49A9-A3FE-A378B03DDB4D}
[...]	

```
{012616AB-FF6D-4503-A6F0-EFFD0523ACE6} {012616AB-FF6D-4503-A6F0-EFFD0523ACE6}
[...]
{FF32ADA1-5A4B-583C-889E-A3C027B201F5} {FF32ADA1-5A4B-583C-889E-A3C027B201F5}

The command completed successfully.
```

Listing 6.9: PoC I - Abfrage von einem Prozess zugeordneten Provider (Auszug)

Im Auszug aus der Eingabeaufforderung (Listing 6.9) werden die ETW-Provider aufgelistet, welche Events mit Informationen liefern, die mit dem ausgeführten Prozess mit der ID 23868 in Relation stehen.

Neben den ETW-Providern, die mit ihrem Bezeichner aufgeführt werden, werden auch Provider aufgelistet, die lediglich über eine GUID verfügen. Bei diesen Providern handelt es sich, nach Matt Graeber, um WPP Software Trace Provider oder TraceLogging Provider [Gra19]. Informationen über diese Art von Providern lassen sich nicht über die beschriebenen Werkzeuge (`logman`, `ETWExplorer` oder `WEPEXplorer`) abrufen, da die Manifeste nicht im System registriert sind. Informationen über diese Provider werden bei WPP Software Trace Providern in sog. TMF-Dateien bereitgestellt, die sich aus den Symbol-Dateien (PDB) extrahieren lassen. Im Fall von TraceLogging Providern werden die Provider- und Eventinformationen selbsterklärend mit dem Quellcode von Programmen ausgeliefert [Gra19]. Für die Analyse von Prozessverhalten werden in dieser Vorgehensweise ausschließlich ETW-Provider einbezogen. Mit der Event Tracing Bibliothek lassen sich die, einem Prozess zugeordneten, Provider über den Funktionsaufruf `TraceEventProviders.GetRegisteredProvidersInProcess(int ProcessID)` als Liste abrufen. Als Rückgabe wird eine Übersicht der ETW-Provider mit Namen und GUID geliefert, die in Relation zum ausgeführten Prozess stehen. Der Prozess aus dem exemplarischen Anwendungsfall wird hierzu gestartet. Bevor dieser mit Ausführung des Testverhaltens fortfährt, ist eine Benutzereingabe erforderlich.

Wird die Abfrage der prozessbezogenen Provider mittels `logman`-Befehlsaufruf vor und nach Durchlauf der Testverhaltensweisen TV1-TV4 auf den Beispielprozess angewendet, ist feststellbar, dass die Anzahl der Provider zum jeweiligen Abfragezeitpunkt abweicht (Listing 6.10 und 6.11):

Provider	GUID
Microsoft-IEFRAME	{5C8BB950-959E-4309-8908-67961A1205D5}
Microsoft-Windows-AppModel-Runtime	{F1EF270A-0D32-4352-BA52-DBAB41E1D859}
Microsoft-Windows-AsynchronousCausality	{19A4C69A-28EB-4D4B-8D94-5F19055A1B5C}
Microsoft-Windows-COM-Perf	{B8D6861B-D20F-4EEC-BBAE-87E0DD80602B}
Microsoft-Windows-COM-RundownInstrumentation	{2957313D-FCAA-5D4A-2F69-32CE5F0AC44E}
Microsoft-Windows-Eventlog	{FC65DDD8-D6EF-4962-83D5-6E5CFE9CE148}
Microsoft-Windows-Heap-Snapshot	{901D2AFA-4FF6-46D7-8D0E-53645E1A47F5}
Microsoft-Windows-Kernel-AppCompat	{16A1ADC1-9B7F-4CD9-94B3-D8296AB1B130}

Microsoft-Windows-KnownFolders	{8939299F-2315-4C5C-9B91-ABB86AA0627D}
Microsoft-Windows-Networking-Correlation	{83ED54F0-4D48-4E45-B16E-726FFD1FA4AF}
Microsoft-Windows-Shell-Core	{30336ED4-E327-447C-9DE0-51B652C86108}
Microsoft-Windows-URLMon	{245F975D-909D-49ED-B8F9-9A75691D6B6B}
Microsoft-Windows-User-Diagnostic	{305FC87B-002A-5E26-D297-60223012CA9C}
Microsoft-Windows-WinRT-Error	{A86F8471-C31D-4FBC-A035-665D06047B03}
{012616AB-FF6D-4503-A6F0-EFFD0523ACE6}	{012616AB-FF6D-4503-A6F0-EFFD0523ACE6}
[...]	
{FF32ADA1-5A4B-583C-889E-A3C027B201F5}	{FF32ADA1-5A4B-583C-889E-A3C027B201F5}

Listing 6.10: PoC I - Abfrage der zugeordneten Provider vor TV1

In Listing 6.11 wird das Ergebnis des logman-Exports nach Durchlauf aller Testverhaltensweisen aufgeführt:

Provider	GUID
-----	-----
Microsoft-IEFRAME	{5C8BB950-959E-4309-8908-67961A1205D5}
Microsoft-Windows-AppModel-Runtime	{F1EF270A-0D32-4352-BA52-DBAB41E1D859}
Microsoft-Windows-AsynchronousCausality	{19A4C69A-28EB-4D4B-8D94-5F19055A1B5C}
Microsoft-Windows-CAPI2	{5BBCA4A8-B209-48DC-A8C7-B23D3E5216FB}
Microsoft-Windows-COM-Perf	{B8D6861B-D20F-4EEC-BBAE-87E0DD80602B}
Microsoft-Windows-COM-RundownInstrumentation	{2957313D-FCAA-5D4A-2F69-32CE5F0AC44E}
Microsoft-Windows-Crypto-BCrypt	{C7E089AC-BA2A-11E0-9AF7-68384824019B}
Microsoft-Windows-Crypto-RSAEnh	{152FDB2B-6E9D-4B60-B317-815D5F174C4A}
Microsoft-Windows-Eventlog	{FC65DDD8-D6EF-4962-83D5-6E5CFE9CE148}
Microsoft-Windows-Heap-Snapshot	{901D2AFA-4FF6-46D7-8D0E-53645E1A47F5}
Microsoft-Windows-Kernel-AppCompat	{16A1ADC1-9B7F-4CD9-94B3-D8296AB1B130}
Microsoft-Windows-KnownFolders	{8939299F-2315-4C5C-9B91-ABB86AA0627D}
Microsoft-Windows-Networking-Correlation	{83ED54F0-4D48-4E45-B16E-726FFD1FA4AF}
Microsoft-Windows-RPC	{6AD52B32-D609-4BE9-AE07-CE8DAE937E39}
Microsoft-Windows-RPC-Events	{F4AED7C7-A898-4627-B053-44A7CAA12FCD}
Microsoft-Windows-Shell-Core	{30336ED4-E327-447C-9DE0-51B652C86108}
Microsoft-Windows-URLMon	{245F975D-909D-49ED-B8F9-9A75691D6B6B}
Microsoft-Windows-User-Diagnostic	{305FC87B-002A-5E26-D297-60223012CA9C}
Microsoft-Windows-UxTheme	{422088E6-CD0C-4F99-BD0B-6985FA290BDF}
Microsoft-Windows-WinHttp	{7D44233D-3055-4B9C-BA64-0D47CA40A232}
Microsoft-Windows-WinINet	{43D1A55C-76D6-4F7E-995C-64C711E5CAFE}
Microsoft-Windows-WinINet-Capture	{A70FF94F-570B-4979-BA5C-E59C9FEAB61B}
Microsoft-Windows-WinRT-Error	{A86F8471-C31D-4FBC-A035-665D06047B03}
Microsoft-Windows-Winsock-NameResolution	{55404E71-4DB9-4DEB-A5F5-8F86E46DDE56}
TCPIP Service Trace	{EB004A05-9B1A-11D4-9123-0050047759BC}
{012616AB-FF6D-4503-A6F0-EFFD0523ACE6}	{012616AB-FF6D-4503-A6F0-EFFD0523ACE6}
[...]	
{FF32ADA1-5A4B-583C-889E-A3C027B201F5}	{FF32ADA1-5A4B-583C-889E-A3C027B201F5}

Listing 6.11: PoC I - Abfrage der zugeordneten Provider nach TV4

Im Beispielprozess wird daher vor und nach jedem Testverhaltensblock logman mit den Parametern query providers -PID 23868 aufgerufen, um zu erfassen, zu welchem Zeitpunkt ETW-Provider mit den Aufrufen der Systemprogrammierschnittstellen innerhalb der Testverhaltensweisen korrelieren (Listing 6.12). Hierdurch

können zusätzlich Rückschlüsse auf Provider geschlossen werden, die mit bestimmten Bibliotheken in Verbindung stehen, bzw. in denen Events geschrieben werden.

```

1 system(("logman.exe query providers -pid " + pid.str() + "> START_TVn_providers.txt").c_str());
2 [...]
3 system(("logman.exe query providers -pid " + pid.str() + "> ENDE_TVn_providers.txt").c_str());

```

Listing 6.12: PoC I - Abfrage zugeordneter Provider zur Laufzeit des Beispielprozesses

Durch Abfrage der prozessbezogenen Provider des DemoProcess zur Laufzeit zwischen den Testverhaltensweisen (siehe Listing 6.12) konnte ermittelt werden, dass

- nach TV1, TV3 und TV4 keine zusätzlichen Provider zugeordnet wurden,
- und nach TV2 weitere zusätzlichen Provider zugeordnet wurden (Listing 6.13).

FWPUCLNT Trace Provider	{5A1600D2-68E5-4DE7-BCF4-1C2D215FE0FE}
Microsoft-Windows-ASN1	{D92EF8AC-99DD-4AB8-B91D-C6EBA85F3755}
Microsoft-Windows-CAPI2	{5BBCA4A8-B209-48DC-A8C7-B23D3E5216FB}
Microsoft-Windows-Crypto-BCrypt	{C7E089AC-BA2A-11E0-9AF7-68384824019B}
Microsoft-Windows-Crypto-NCrypt	{E8ED09DC-100C-45E2-9FC8-B53399EC1F70}
Microsoft-Windows-Crypto-RSAEnh	{152FDB2B-6E9D-4B60-B317-815D5F174C4A}
Microsoft-Windows-DNS-Client	{1C95126E-7EEA-49A9-A3FE-A378B03DDB4D}
Microsoft-Windows-RPC	{6AD52B32-D609-4BE9-AE07-CE8DAE937E39}
Microsoft-Windows-RPC-Events	{F4AED7C7-A898-4627-B053-44A7CAA12FCD}
Microsoft-Windows-Schannel-Events	{91CC1150-71AA-47E2-AE18-C96E61736B6F}
Microsoft-Windows-Shell-Core	{30336ED4-E327-447C-9DE0-51B652C86108}
Microsoft-Windows-User-Diagnostic	{305FC87B-002A-5E26-D297-60223012CA9C}
Microsoft-Windows-UxTheme	{422088E6-CD0C-4F99-BD0B-6985FA290BDF}
Microsoft-Windows-WebIO	{50B3E73C-9370-461D-BB9F-26F32D68887D}
Microsoft-Windows-WinHttp	{7D44233D-3055-4B9C-BA64-0D47CA40A232}
Microsoft-Windows-WinINet	{43D1A55C-76D6-4F7E-995C-64C711E5CAFE}
Microsoft-Windows-WinINet-Capture	{A70FF94F-570B-4979-BA5C-E59C9FEAB61B}
Microsoft-Windows-WinRT-Error	{A86F8471-C31D-4FBC-A035-665D06047B03}
Microsoft-Windows-Winsock-NameResolution	{55404E71-4DB9-4DEB-A5F5-8F86E46DDE56}
Security: SChannel	{37D2C3CD-C5D4-4587-8531-4696C44244C8}
TCPIP Service Trace	{EB004A05-9B1A-11D4-9123-0050047759BC}

Listing 6.13: PoC I - Abfrage der zugeordneten Provider nach TV2

Neben den registrierten Providern des User-Modes lassen sich Provider innerhalb des Kernel-Mode (Built-in Kernel Provider) mittels Keywords (Enable Flags) aktivieren [PB15]. Der Provider Windows Kernel Trace stellt Events aus dem Kernel zur Verfügung. Diese lassen sich nur von der NT Kernel Logger Sitzung empfangen [Sou12, S. 419f]. Eine Auflistung der Keywords kann mittels logman (siehe Listing 6.14) vorgenommen werden. Im Anhang C.1 werden die Kernel Keywords der ETW-API der TraceEvent Library gegenübergestellt.

```

C:\Windows\system32>logman query providers "Windows Kernel Trace"
Value           Keyword           Description
-----
0x0000000000000001  process           Process creations/deletions

```

0x0000000000000002	thread	Thread creations/deletions
0x0000000000000004	img	Image load
0x0000000000000008	procctr	Process counters
0x0000000000000010	cswitch	Context switches
0x0000000000000020	dpc	Deferred procedure calls
0x0000000000000040	isr	Interrupts
0x0000000000000080	syscall	System calls
0x0000000000000100	disk	Disk IO
0x0000000000000200	file	File details
0x0000000000000400	diskinit	Disk IO entry
0x0000000000000800	dispatcher	Dispatcher operations
0x0000000000001000	pf	Page faults
0x0000000000002000	hf	Hard page faults
0x0000000000004000	virtualloc	Virtual memory allocations
0x0000000000010000	net	Network TCP/IP
0x0000000000020000	registry	Registry details
0x0000000000100000	alpc	ALPC
0x0000000000200000	splitio	Split IO
0x0000000000800000	driver	Driver delays
0x0000000001000000	profile	Sample based profiling
0x0000000002000000	fileiocompletion	File IO completion
0x0000000004000000	fileio	File IO

Listing 6.14: Abfrage der Kernel Keywords des Windows Kernel Traces

6.1.4 Aufzeichnen des Prozessverhaltens

Erstellen einer ETW-Sitzung für die Analyse (ETW-Analysesitzung)

Programmiertechnisch wird eine ETW-Sitzung im Analysewerkzeug durch ein Objekt der Klasse `TraceEventSession` der Event Tracing API repräsentiert. Über die Klasse `TraceEventSession` wird daher ein Objekt für die Analysesitzung instanziiert (Listing 6.15), auf denen die sitzungsspezifischen Operationen ausgeführt werden:

```
1 TraceEventSession session = new TraceEventSession ("ETWProcessTracerSession");
```

Listing 6.15: Instanziierung des `TraceEventSession` Objekts für die Analysesitzung

Im erstellten Sitzungsobjekt können sowohl User-Mode Provider als auch die Provider im Kernel über die NT Kernel Logger Session aufgenommen werden.

Zuordnen von ETW-Providern

Die Auswahl der in die Analyse einzubeziehenden Providern ist vom Kommandozeilenparameter des Analysewerkzeuges abhängig. Es wird zwischen zwei möglichen Varianten unterschieden:

In der ersten Variante (Standard) bezieht das Analysewerkzeug automatisch alle im System registrierten ETW-Provider über die Trace Event Bibliothek, während in der alternativen Variante eine Liste von individuell für die Analyse ausgewählten ETW-Provider eingelesen werden kann. Bei der ersten Variante ist kein zusätzlicher Konsolenparameter notwendig. Über den Funktionsaufruf `GetPublishedProviders()` des Objektes `TraceEventProvider` der Event Tracing Bibliothek werden alle im System registrierten ETW-Provider ermittelt und der Analysesitzung (ETWProcessTracerSession) zugeordnet. Bei der Aktivierung eines ETW-Providers wird zusätzlich das Event Log Level „Verbose“ festgelegt, um alle Events zu erfassen.

Durch ein Exception-Handling werden mögliche nicht aktivierbare Provider abgefangen. Können Provider nicht aktiviert werden, wird dies auf der Konsolenausgabe vermerkt (Listing 6.16):

```

1 // Add all system registered providers (User-Mode) to the ETWProcessTracer session
2
3 // Try enable any registered provider of the current system to the ETWProcessTracer session
4 foreach (var provider in TraceEventProviders.GetPublishedProviders()) {
5     try
6     {
7         session.EnableProvider(provider, TraceEventLevel.Verbose, 0);
8
9         // Write to console on success
10        Console.WriteLine("Provider: " + TraceEventProviders.GetProviderName(provider) + " enabled.");
11    }
12    catch
13    {
14        // Write to console in case of a provider could not be enabled to the session
15        Console.WriteLine("Provider: " + TraceEventProviders.GetProviderName(provider) + " could not
16        be enabled.");
17    }
18 }

```

Listing 6.16: Zuordnung der im System registrierten Provider zur Analysesitzung

In der zweiten Variante wird dem Analysewerkzeug eine Liste von einzubeziehenden ETW-Providern übergeben (Aufrufparameter: `--ProviderConfig [Pfad zur Listendatei (TXT)]`). Die Liste muss die Form: „ProviderName {GUID}“ aufweisen; wie sie auch im Werkzeug `logman` dargestellt wird. Die Liste wird durch das Analysewerkzeug als Textdatei eingelesen. In einer Schleife werden alle Provider der Liste zur Analysesitzung zugeordnet. Die programmiertechnische Umsetzung erfolgt analog zur Darstellung in Listing 6.16.

Da es sich beim NT Kernel Logger um eine im System vordefinierte Sitzung handelt, der bereits verschiedene Provider aus dem Kernel zugeordnet sind, ist ein Aktivieren und Zuordnen von ETW-Providern, wie in Listing 6.16 dargestellt, nicht möglich.

Stattdessen werden einzelne Event-Kategorien über die Kernel-Keywords der Kernel-TraceEventParser konfiguriert. Das Keyword „All“ umfasst gemäß Dokumentation der Event Tracing Bibliothek die Kernel-Keywords:

AdvancedLocalProcedureCalls, DeferedProcedureCalls, ContextSwitch, DiskFileIO, DiskIO, DiskIOInit, Dispatcher, Driver, FileIO, FileIOInit, ImageLoad, Interrupt, Memory, MemoryHardFaults, NetworkTCPIP, Process, ProcessCounters, Profile, Registry, SplitIO, SystemCall, Thread, VAMap und VirtualAlloc.

Im Anhang C wird über die Tabelle C.1 eine Zuordnung der Kernel-Keywords der ETW-API mit denen der Event Tracing Bibliothek vorgenommen. Die Tabelle enthält Entwicklerkommentare zum Informationsgehalt der einzelnen Kernel Keywords sowie der zu erwartenden Frequenz der gelieferten Events.

Die Provider dieser speziellen Sitzung sind im Kernel sowie in Kernel-Mode Treibern implementiert und können Events nicht an individuell erstellte Sitzungen liefern. Die einzelnen Kernel-Provider sind nicht aus dem User-Mode auflistbar. Daher ist die Filterung lediglich auf Basis der Keywords möglich. Um die aufzuzeichnenden Events zu begrenzen, lässt sich mit dem Parameter `--KernelKeywords` im `ETWProcessTracer` festlegen, dass beispielsweise lediglich das Keyword *Process* aktiviert wird:

```
1 session.EnableKernelProvider(KernelTraceEventParser.Keywords.Process);
```

Listing 6.17: Zuordnung des Kernel Traces zur Analysesitzung (nur Process)

Mit der festgelegten Option werden prozessbezogene Ereignisse wie das Starten und Beenden von Prozessen miteinbezogen. Einige der in diesem Fall nicht erfassten Events aus dem Kernel, lassen sich über die ETW-Provider im User-Mode erfassen. Im Analysewerkzeug werden programmiertechnisch alle Ereignisse aus dem Kernel-Provider unabhängig speziell ausgewählter Keywords über einen `KernelTraceEventParser` abgerufen, wenn die Option „All“ gewählt wird (Listing 6.18):

```
1 session.EnableKernelProvider(KernelTraceEventParser.Keywords.All);
```

Listing 6.18: Zuordnung des Kernel Traces zur Analysesitzung (alle Keywords)

Konfiguration der Aufzeichnungsmethodik

Die Events aus den Providern des User- als auch Kernel-Mode werden durch die Analysesitzung in Echtzeit aufgezeichnet und durch die Sitzungsobjekte der Klasse `TraceEventSession` repräsentiert. Mit Hilfe von Callbacks wird für jedes gelieferte Event eine Programmroutine abgearbeitet, in der zunächst geprüft wird, ob die zu beobachtenden Prozesse (in der Watchlist) im Event-Header zugeordnet werden können (Listing 6.19):

```

1 // Set Trace Event Callbacks for User and Kernel Trace
2
3 // User Trace - Callback (All Events)
4 session.Source.Dynamic.All += delegate (TraceEvent data) {
5 // Check if ProcessID of event header matches the watchlist
6     if (pidWatchlist.IndexOf(data.ProcessID) != -1) {
7         // Increase eventCounter
8         eventCounter++;
9
10        // Prepare event data and mark event for CSV export
11        csvFile.AppendLine(prepareEvent(data));
12
13        // Update eventCounter on command line
14        printEventCounter();
15    }
16 };
17 [...]
18 // Kernel Trace - Callback (All Events)
19 session.Source.Kernel.All += delegate (TraceEvent data) {
20 // Check if ProcessID of event header matches the watchlist
21     if (pidWatchlist.IndexOf(data.ProcessID) != -1) {
22         // Increase eventCounter
23         eventCounter++;
24
25        // Prepare event data and mark event for CSV export
26        csvFile.AppendLine(prepareEvent(data));
27
28        // Update eventCounter on command line
29        printEventCounter();
30    }
31 };

```

Listing 6.19: Filterung prozessbezogener Events mit Hilfe von Callbacks

Ist ein Prozess in der Liste der zu überwachenden PIDs im Event-Header enthalten, wird das entsprechende Event für einen Export in der CSV-Protokolldatei vorge-merkt. Das Parsen der Event-Strukturen wird durch die Trace Logging Bibliothek automatisch im XML-Stil vorgenommen.

Für den Prozessstart wird ein gesonderter Callback des Kernel Traces eingerichtet (siehe Listing 6.20), um die Liste der zu überwachenden Prozess-IDs (Watchlist) zur Laufzeit der Analyse zu aktualisieren:

```

1 // Kernel Trace Callback for Process START events (of spawned processes)
2 session.Source.Kernel.ProcessStart += delegate (ProcessTraceData data) {
3 // Check if ProcessID of event header matches the watchlist
4     if (pidWatchlist.IndexOf(data.ParentID) != -1) {
5         // Add new PID of spawned process to the Watchlist
6         pidWatchlist.Add(data.ProcessID);
7         [...]
8     }
9 };
10 [...]

```



```

11 // Kernel Trace Callback for Process STOP events
12 session.Source.Kernel.ProcessStop += delegate (ProcessTraceData data) {
13 // Check if ProcessID of event header matches the watchlist
14     if (pidWatchlist.IndexOf(data.ProcessID) != -1) {
15         // Remove PID of terminated process from Watchlist
16         pidWatchlist.Remove(data.ProcessID);
17         [...]
18     }
19 };

```

Listing 6.20: Erfassen von neu erstellten (Kind-)Prozessen durch Kernel-Trace Callbacks

Durch diesen Callback werden alle erzeugten Kindprozesse des von der Analyse ausgehenden Prozesses mitverfolgt. Nach Terminierung eines Prozesses wird die zugeordnete PID aus der Analyse entfernt, da das Betriebssystem die PID nach Prozessterminierung erneut vergeben könnte und die Analyseergebnisse verfälscht.

Starten der Aufzeichnung durch Aktivieren der ETW-Sitzungen

Für den Start der Aufzeichnung der ETW-Analysesitzung ist die Angabe einer PID erforderlich, zu der die zugeordneten Events aus dem Event-Datenstrom der Provider herausgefiltert werden. Im ersten Schritt wird mit Hilfe des `ProcessStartHelper` der Ladevorgang des Beispielprozesses (`DemoProcess`) vorbereitet (Listing 6.21):

```

C:\> ProcessStartHelper.exe DemoProcess.exe

ProcessStartHelper
PID: 4828

Press any key for loading DemoProcess.exe as suspended process.

```

Listing 6.21: PoC I - Laden des `DemoProcess` mit dem `ProcessStartHelper`

Die bereitgestellte PID des `ProcessStartHelper` (4828) wird für den Aufruf des `ETWProcessTracer` als Parameter `--PID` angegeben (Listing 6.22):

```

C:\> ETWProcessTracer.exe --PID 4828 --KernelKeywords All
[...]
CONFIGURATION:
-----
Target ProcessID:      4828
ProviderConfig:       all registered ETW-Providers
KernelKeywords:       All

Logfile directory:    C:\etw\ETWProcessTracer\
-----
[INFO] Press CTRL+C at any time to stop tracing
Press <Enter> to start monitoring...

```

Listing 6.22: PoC I - Start der Analyse mit dem `ETWProcessTracer`

Nach Überprüfung der Konfiguration der Analysesitzung wird die ETW-Sitzung und Aufzeichnung der Events durch Benutzereingabe gestartet. Alle im System registrierten Provider werden aktiviert und die PID 4828 als initial zu überwachender Prozess in die Liste aufgenommen.

Prozess starten und überwachen

Der bereits vorbereitete `ProcessStartHelper` wird durch Benutzereingabe zum Laden des Beispielprozesses aufgefordert (Listing 6.23):

```
C:\> ProcessStartHelper.exe DemoProcess.exe

ProcessStartHelper
PID: 4828

Press any key for loading DemoProcess.exe as suspended process.

OK: DemoProcess.exe assigned ProcessID: 416 [SUSPENDED]
Press <ENTER> to resume process
```

Listing 6.23: PoC I - Start des DemoProcess mit dem ProcessStartHelper

Die PID des `DemoProcess` wird durch das Analysewerkzeug als erzeugter Kindprozess in die Liste der zu überwachenden Prozesse aufgenommen und entsprechend zur Filterung der Events aus den aktivierten Providern über die Callbacks herangezogen. Über die Konsolenausgabe des `ETWProcessTracers` können die erstellten und terminierenden Prozesse überwacht werden (Listing 6.24):

```
[PROCESS CREATED] PID: 416 (PPID: 4828) ProcessName: "DemoProcess" ImagePath: "
  DemoProcess.exe" Args: "DemoProcess.exe"
[PROCESS TERMINATED] PID: 4828 (PPID: 7040) ProcessName: "" ImagePath: "
  ProcessStartHelper.exe" Args: "ProcessStartHelper.exe DemoProcess.exe"
[PROCESS CREATED] PID: 5464 (PPID: 416) ProcessName: "winver" ImagePath: "winver.
  exe" Args: "C:\Windows\System32\winver.exe"
[PROCESS TERMINATED] PID: 5464 (PPID: 416) ProcessName: "" ImagePath: "winver.exe"
  Args: "C:\Windows\System32\winver.exe"
[PROCESS TERMINATED] PID: 416 (PPID: 4828) ProcessName: "" ImagePath: "DemoProcess.
  exe" Args: "DemoProcess.exe"
```

Listing 6.24: PoC I - Erfassung des gestarteten DemoProcess im ETWProcessTracer

```
4828: ProcessStartHelper (Args: "ProcessStartHelper.exe DemoProcess.exe")
├── 416: DemoProcess ("DemoProcess.exe")
│   └── 5464 : winver (Args: "C:\Windows\System32\winver.exe")
```

Bild 6.2: PoC I - Prozessbaum über den DemoProcess

Nachdem der `DemoProcess` durch den `ProcessStartHelper` gestartet wurde, terminiert der zugehörige Prozess mit der ID 4828.

Aufzeichnung beobachten und Events filtern

Werden Events aus einem bestimmten Provider geliefert, wird eine Callback-Operation ausgelöst, die das entsprechende Event behandelt und weiterverarbeitet. Hierbei werden insbesondere die bereits beschriebenen Filterregeln nach den PIDs angewendet. Auf der Konsole werden, neben den neu erstellten (Kind-)Prozessen des initialen Prozesses, auch die Anzahl bereits aufgezeichneter Events ausgegeben.

Stoppen der Aufzeichnung nach Terminierung des Prozesses

Terminiert der Beispielprozess, kann die Aufzeichnung gestoppt werden. Über die Konsolenausgabe erfolgt eine Benachrichtigung (Listing 6.25):

```
[PROCESS TERMINATED] PID: 416 (PPID: 4828) ProcessName: "" ImagePath: "DemoProcess.exe" Args: ""DemoProcess.exe""
```

Listing 6.25: PoC I - Erfassung des terminierten `DemoProcess` im `ETWProcessTracer`

Durch die Tastenkombination `STRG+C` (siehe Listing 6.26) wird die Analysesitzung aufgeräumt und beendet. Für das Sitzungsobjekt wird dazu programmintern die `Dispose()`-Funktion aufgerufen, um die ETW-Sitzung ordnungsgemäß zu beenden.

```
[CTRL+C] ETWProcessTracerSession will be terminated. Please wait...
```

Listing 6.26: PoC I - Information über das Ende der Aufzeichnung

6.1.5 Auswertung der Aufzeichnung

Nach Abschluss der Aufzeichnung wird die Protokolldatei (CSV-Datei) aus dem Ausgabeverzeichnis des Analysesystems abgerufen. Sie verfügt über die dekodierten Events und kann daher als lesbares Format beispielsweise in einer Tabellenkalkulationssoftware oder dem DB4S (DB Browser for SQLite) weiterverarbeitet und ausgewertet werden. Zur Anwendung von Filterregeln wurde die in Tabelle 6.1 dargestellte Spaltenstruktur gewählt:

Tabelle 6.1: Kopfzeile (Header) der CSV-Protokolldatei des `ETWProcessTracers`

ID	Time	PID	ProviderName	EventName	EventMessage
----	------	-----	--------------	-----------	--------------

Für die Auswertung werden die Spalten nach Trennzeichen (hier: Semikolon) automatisch gefiltert. In der Protokolldatei werden nur solche Provider aufgelistet, aus denen tatsächlich Events aufgezeichnet wurden. Durch die Filterung der Spalte *ProviderName* lassen sich die Provider ermitteln, welche Events lieferten die Rückschlüsse auf das jeweilige Testverhalten gegeben haben. Die Spalten *ProviderName* und *EventName* erleichtern eine spätere Auswertung. Aufgezeichnete Events der NT Kernel Logger Sitzung lassen sich nur nach *EventName* kategorisieren, da der *ProviderName* keine Unterscheidung nach Keywords vornimmt und einheitlich als Windows Kernel oder MSNT_SystemTrace dargestellt wird.

Der Prozessstart des Beispielprozesses wird aus dem Kernel Provider über das Event `Process/Start` identifiziert (1), auf welches das Events `Image/Load` (2) folgt und bestätigt, dass das Image der ausführbaren Datei geladen wird. Aus den Events `ProcessStart/Start` (3) sowie `ImageLoad` (4) aus dem Provider `Microsoft-Windows-Kernel-Process` wird Prozessstart des `DemoProcess` zusätzlich protokolliert, dass das zugehörige Image `DemoProcess.exe` geladen wurde:

```

1 <Event MSec= "1891,8057" PID= "416" PName="DemoProcess" TID= "-1" EventName="Process/Start"
  ProcessID="416" ParentID="4.828" ImageFileName="DemoProcess.exe"
  DirectoryTableBase="0x96DB6000" Flags="None" SessionID="1" ExitStatus="0x0000103"
  UniqueProcessKey="0xFFFFA903DC39F080" CommandLine=" DemoProcess.exe "/>
2 <Event MSec= "4635,5259" PID= "416" PName="DemoProcess" TID="4056" EventName="Image/Load"
  ImageBase="0x00007FF7BDEA0000" ImageSize="0x00057000" ImageChecksum="0" TimeDateStamp=
  "1.596.030.546" DefaultBase="0x00007FF7BDEA0000" FileName="C:\etw\DemoProcess.exe"/>
3 <Event MSec= "1891,8017" PID="4828" PName="ProcessStartHelper" TID="1804"
  EventName="ProcessStart/Start" ProviderName="Microsoft-Windows-Kernel-Process"
  FormattedMessage="Process 416 started at time 16:50:24.856966 (1.891,691 MSec) by parent 4.828
  running in session 1 with name &lt; &lt; BadFieldIdx&gt; &gt; . " ProcessID="416"
  ProcessSequenceNumber="1.005" CreateTime="16:50:24.856966 (1.891,691 MSec)"
  ParentProcessID="4.828" ParentProcessSequenceNumber="1.002" SessionID="1" Flags="0"
  ProcessTokenElevationType="3" ProcessTokenIsElevated="0"/>
4 <Event MSec= "4635,5208" PID= "416" PName="DemoProcess" TID="4056" EventName="ImageLoad"
  ProviderName="Microsoft-Windows-Kernel-Process" FormattedMessage="Process 416 had an image
  loaded with name \Device\HarddiskVolume2\etw\DemoProcess.exe. " ImageBase="0x7ff7bdea0000"
  ImageSize="0x00057000" ProcessID="416" ImageChecksum="0" TimeDateStamp="1.596.030.546"
  DefaultBase="0x7ff7bdea0000" ImageName="\Device\HarddiskVolume2\etw\DemoProcess.exe"/>

```



Im Folgenden werden auszugsweise die relevanten Events (nur EventMessage der CSV-Datei) aus der aufgezeichneten ETW-Analysesitzung aufgeführt, die dem einzelnen Testverhalten im exemplarischen Anwendungsfall zugeordnet werden konnten. Aufgezeichnete Events aus den Providern der NT Kernel Logger Sitzung verfügen über Schnittmengen mit Events aus den User-Mode Providern `Microsoft-Windows-Kernel-*`. Der Informationsgehalt unterscheidet sich zwischen den verschiedenen Event-Kategorien geringfügig.

TV1: Auslesen der Registry-Werte

Für das Testverhalten 1 konnten die Events über das Prozessverhalten aus den Providern:

- Microsoft-Windows-Kernel-Registry
- Windows Kernel (Keywords: Registry)

bezogen werden.

Die 1. Registry-Abfrage zur Existenz des Schlüssels "VirtualBox Guest Additions" lässt sich neben dem ETW-Provider Microsoft-Windows-Kernel-Registry (5), als auch aus dem Kernel Provider (6) entnehmen:

```

5 <Event MSec= "13381,4743" PID= "416" PName="DemoProcess" TID="4056" EventName="EventID(2) "
  ProviderName="Microsoft-Windows-Kernel-Registry" BaseObject="0xffffba08578e2a70"
  KeyObject="0x00000000" Status="-1.073.741.772" Disposition="0" BaseName=""
  RelativeName="SOFTWARE\Oracle\VirtualBox Guest Additions"/>
6 <Event MSec= "13381,4768" PID= "416" PName="DemoProcess" TID="4056" EventName="Registry/Open"
  Status="0x00000000" KeyHandle="0xFFFFFBA084A843EC0" ElapsedTimeMSec="0,0179000000007363"
  KeyName="SOFTWARE\Oracle\VirtualBox Guest Additions" Index="0"/>

```

In der 2. Registry-Abfrage wurde der Schlüsselwert (SystemBiosVersion) im Schlüssel HKLM\HARDWARE\Description\System ermittelt. Zunächst wird durch den Demo Process der Registrierungsschlüsselpfad, analog zur 1. Abfrage geöffnet (7). Analog zum Event mit der ID 2 (OpenKey) des Providers Microsoft-Windows-Kernel-Registry, wird ein Registry/Open-Event aus dem Windows Kernel aufgezeichnet (8):

```

7 <Event MSec= "14414,1157" PID= "416" PName="DemoProcess" TID="4056" EventName="EventID(2) "
  ProviderName="Microsoft-Windows-Kernel-Registry" BaseObject="0xffffba08578e2a70"
  KeyObject="0xffffba0857f09970" Status="0" Disposition="0" BaseName=""
  RelativeName="HARDWARE\Description\System"/>
8 <Event MSec= "14414,1240" PID= "416" PName="DemoProcess" TID="4056" EventName="Registry/Open"
  Status="0x00000000" KeyHandle="0xFFFFFBA084A843EC0" ElapsedTimeMSec="0,049999999992724"
  KeyName="HARDWARE\Description\System" Index="0"/>

```

Festzustellen ist, dass der abgefragte Schlüsselwertname im Event mit der ID 7 (QueryValueKey) aus dem Provider Microsoft-Windows-Kernel-Registry im Feld ValueName (SystemBiosVersion) durch das Event geliefert wird (9), nicht jedoch der Wert des Schlüssels. Über den Kernel Provider wird weder der Schlüsselwertname noch der Wert des Schlüssels im Registry/QueryValue-Event (10) zurückgeliefert.

```

9 <Event MSec= "14414,1586" PID= "416" PName="DemoProcess" TID="4056" EventName="EventID(7) "
  ProviderName="Microsoft-Windows-Kernel-Registry" KeyObject="0xffffba0857f09970" Status="0"
  InfoClass="2" DataSize="36" KeyName="" ValueName="SystemBiosVersion" CapturedData=""/>
10 <Event MSec= "3657,7889" PID="416" PName="DemoProcess" TID="21304" EventName="Registry/QueryValue"
  Status="0x00000000" KeyHandle="0xFFFFFBA095E5F2820" ElapsedTimeMSec="0,0054000000000873"
  KeyName="" Index="0"/>

```

Neben dem Auslesen der Registrierungsschlüssel und -werten wurde der Registry-Schlüsselwertname TV1DemoRunKey mit Wert C:\TV1DemoRunKey.exe angelegt (11). Auch hier zeigen die Events ein ähnliches Verhalten wie in der 2. Abfrage. Der Schüsselpfad ist vollständig aus den zugehörigen Events ablesbar (12, 13), während der gesetzte Schlüsselwert über das Event Registry/SetValue nicht entnommen werden kann (14):

```

11 <Event MSec= "16476,8247" PID= "416" PName="DemoProcess" TID="4056" EventName="EventID(2) "
    ProviderName="Microsoft-Windows-Kernel-Registry" BaseObject="0xffffba0857f09770"
    KeyObject="0xffffba0857f09770" Status="0" Disposition="0" BaseName=""
    RelativeName="SOFTWARE\Microsoft\Windows\CurrentVersion\Run" />
12 <Event MSec= "16476,8322" PID= "416" PName="DemoProcess" TID="4056" EventName="Registry/Open"
    Status="0x00000000" KeyHandle="0xFFFFBA084F144DF0" ElapsedTimeMSec="0,0380000000004657"
    KeyName="SOFTWARE\Microsoft\Windows\CurrentVersion\Run" Index="0" />
13 <Event MSec= "16478,0000" PID= "416" PName="DemoProcess" TID="4056" EventName="EventID(5) "
    ProviderName="Microsoft-Windows-Kernel-Registry" KeyObject="0xffffba0857f09770" Status="0"
    Type="1" DataSize="42" KeyName="" ValueName="TV1DemoRunKey" CapturedData=""
    PreviousDataType="0" PreviousDataSize="0" PreviousData="" />
14 <Event MSec= "5660,8586" PID="416" PName="DemoProcess" TID="21304" EventName="Registry/SetValue"
    Status="0x00000000" KeyHandle="0xFFFFAB095E5F3BD0" ElapsedTimeMSec="0,040999999992579"
    KeyName="" Index="0" />

```

TV2: Nachladen einer Datei von einem Webserver

Für das Testverhalten 2 konnten die Events über das Prozessverhalten aus den Providern:

- Microsoft-Windows-CAPI2
- Microsoft-Windows-DNS-Client
- Microsoft-Windows-FileInfoMinifilter
- Microsoft-Windows-Kernel-File
- Microsoft-Windows-TCPIP
- Microsoft-Windows-URLMon
- Microsoft-Windows-WebIO
- Microsoft-Windows-WinHTTP
- Microsoft-Windows-WinINet
- Microsoft-Windows-Winsock-AFD
- Microsoft-Windows-Winsock-NameResolution
- Windows Kernel (Keywords: FileIO, NetworkTCPIP)

zugeordnet werden. Auszugsweise werden aus den Ergebnissen aufgezeichnete Events aufgelistet. Im Event URLMON_CInet_Start (15) des Providers Microsoft-Windows-URLMon lässt sich die URL zur angefragten Ressource (favicon_hsw_fiw.png)

entnehmen. Das Event WININET_HTTP_REQUEST_HANDLE_CREATED (16) aus dem Provider Microsoft-Windows-WinINet bestätigt die Erzeugung eines Handles für den HTTP-Request der Ressource. Im Provider Microsoft-Windows-DNS-Client (18, 19) und Microsoft-Windows-Winsock-NameResolution (20) lassen sich Events zur Namensauflösung beobachten:

```

15 <Event MSec= "21560,0708" PID= "416" PName="DemoProcess" TID="4056" EventName="URLMON_CInet_Start"
    ProviderName="Microsoft-Windows-URLMon" Flags="1.048.768"
    URL="https://fiw.hs-wismar.de/storages/hs-wismar/favicons/favicon_hsw_fiw.png"/>
16 <Event MSec= "21599,1142" PID= "416" PName="DemoProcess" TID="4056"
    ActivityID="00cc000c00060000a001d80f70ea43fd" EventName="WININET_HTTP_REQUEST_HANDLE_CREATED"
    ProviderName="Microsoft-Windows-WinINet" FormattedMessage="Request handle 0x00cc000c created
    by HttpOpenRequest: ConnectionHandle=0x00cc0008, GET, Target=/storages/hs-wismar/favicons/
    favicon_hsw_fiw.png, Ver=HTTP/1.1, Referrer=, Media types=, Flags=12.582.928 " Connection
    Handle="0x00cc000c" ParentHandle="0x00cc0008" Verb="GET" ObjectName="/storages/hs-wismar/fav
    icons/favicon_hsw_fiw.png" Version="HTTP/1.1" Referrer="" AcceptTypes="" Flags="12.582.928"/>
17 <Event MSec= "21656,3739" PID= "416" PName="DemoProcess" TID="7580"
    ActivityID="00cc000c00060000a001d80f70ea43fd" EventName="Wininet_ResolveHost/Start"
    ProviderName="Microsoft-Windows-WinINet" Request="0x2a1fd43ea70" HostName="fiw.hs-wismar.de"/>
18 <Event MSec= "21656,4547" PID= "416" PName="DemoProcess" TID="7580"
    ActivityID="00cc000c00060000a001d80f70ea43fd" EventName="WININET_DNS_QUERY/Start"
    ProviderName="Microsoft-Windows-WinINet" FormattedMessage="DNS query for fiw.hs-wismar.de
    hostname is sent: Handle=0x00cc000c " HostName="fiw.hs-wismar.de" RequestHandle="0x00cc000c"/>
19 <Event MSec= "21659,7208" PID= "416" PName="DemoProcess" TID="7580"
    ActivityID="00cc000c00060000a001d80f70ea43fd" EventName="EventID(3006)"
    ProviderName="Microsoft-Windows-DNS-Client" FormattedMessage="DNS query is called for the name
    fiw.hs-wismar.de, type 28, query options 1.073.897.472, Server List , isNetwork query 0,
    network index 0, interface index 0, is asynchronous query 0 " QueryName="fiw.hs-wismar.de"
    QueryType="28" QueryOptions="1.073.897.472" ServerList="" IsNetworkQuery="0"
    NetworkQueryIndex="0" InterfaceIndex="0" IsAsyncQuery="0"/>
20 <Event MSec= "21726,8054" PID= "416" PName="DemoProcess" TID="7580"
    ActivityID="00cc000c00060000a001d80f70ea43fd" EventName="WinsockGai"
    ProviderName="Microsoft-Windows-Winsock-NameResolution" FormattedMessage="NSPLookupServiceNext
    is completed for provider 22059d40-7e9e-11cf-ae5a-00aa00a7112b, control Flags 0 and lookup
    Handle 2.894.762.165.872 with status 0 and result [::ffff:141.53.15.120]:53 "
    ProviderGUID="22059d40-7e9e-11cf-ae5a-00aa00a7112b" ControlFlags="0"
    LookupHandle="2.894.762.165.872" Status="0" Result="[::ffff:141.53.15.120]:53"/>

```

Das AfdConnectExWithAddress/Bound-Event (21) des Providers Microsoft-Windows-Winsock-AFD sowie darauffolgende TcpIp/Connect, Send und Recv-Events des Windows Kernels (22-24) stellen Informationen über den Verbindungsaufbau zur aufgelösten Ziel IP-Adresse 141.53.15.120 auf Port 443 dar. Das Event VerifyRevocation/Stop (25) aus dem Provider Microsoft-Windows-CAPI2 liefert Informationen zum überprüften Webserverzertifikat und der Sperllistenprüfung:

```

21 <Event MSec= "21735,9477" PID= "416" PName="DemoProcess" TID="7580" ActivityID="dd085130a903ffff00
    00000000000000" EventName="AfdConnectExWithAddress/Bound" ProviderName="Microsoft-Windows-
    Winsock-AFD" FormattedMessage="ConnectEx: 0: Process 0xffffa903dc39f080, Endpoint 0xffffa903
    dd085130, Buffer 0xffffa903d88dd200, Length 0, Address 141.53.15.120:443, Seq 5.031, Status 0
    " EnterExit="0" Location="5.031" Process= "0xffffa903dc39f080" Endpoint="0xffffa 903dd085130"
    Buffer="0xffffa903d88dd200" BufferLength="0" Status="0" Address="141.53.15.120:443"/>
22 <Event MSec= "21789,8957" PID= "416" PName="DemoProcess" TID= "-1" EventName="TcpIp/Connect"

```



```

size="0" daddr="141.53.15.120" saddr="10.0.2.15" dport="443" sport="51.295" mss="1.460"
sackopt="0" tsopt="0" wsopt="0" rcvwin="65.535" rcvwinscale="0" sndwinscale="0" seqnum="0"
connid="0x4CA397BC00000000"/>
23 <Event MSec= "21806,4712" PID= "416" PName="DemoProcess" TID= "-1" EventName="TcpIp/Send"
size="177" daddr="141.53.15.120" saddr="10.0.2.15" dport="443" sport="51.295"
starttime="233.159" endtime="233.159" seqnum="0" connid="0x0000000500000000"/>
24 <Event MSec= "21835,6319" PID= "416" PName="DemoProcess" TID= "-1" EventName="TcpIp/Recv"
daddr="141.53.15.120" saddr="10.0.2.15" dport="443" sport="51.295" size="1.420"
connid="0xFFFFFFFF00000000" seqnum="-1"/>
25 <Event MSec= "22311,5365" PID= "416" PName="DemoProcess" TID="7580"
ActivityID="00cc000c00060000a001d80f70ea43fd" EventName="VerifyRevocation/Stop"
ProviderName="Microsoft-Windows-CAPI2" FormattedMessage="For more details for this event,
please refer to the Details section " EventWriteData="CertVerifyRevocation Certificate
fileRef= E7E6DD83B698C40EFA1F0BCA3A2F9F2AF3AE294.cer subjectName= www.hs-wismar.de Issuer
Certificate fileRef= C9DCB047AC8C5F0905ED77528CBD4B84D9463C45.cer subjectName= DFN-Verein
Global Issuing CA /&gt; &lt; Flags value= 4 CERT_VERIFY_REV_ACCUMULATIVE_TIMEOUT_FLAG= true /&gt;
&lt; AdditionalParameters currentTime= 2020-07-13T 14:50:44.924Z /&gt; &lt; RevocationStatus
index= 0 error= 0 reason= 0 actualFreshnessTime= PT0S thirdPartyProviderUsed= C:\Windows\
System32\cryptnet.dll /&gt; &lt; OCSPResponse location= Wire url= http://ocsp.pca.dfn.de
/OCSP-Server/OCSP/ME0wSzBJMEcwrTAJBgUrDgMCGGUABBSmAdwSbpEyVgcERAVbBJUJZdQngQUazqYi%2FnyU4na4K2y
Mh4JH%2Biq03QCDB9a6F895VLroynGw%3D%3D fileRef= EEA7423E0EFEFA36FFD43AC7636960EC99CD9B34.bin
issuerName= DFN-Verein Global Issuing CA /&gt; &lt; EventAuxInfo ProcessName= DemoProcess.exe
/&gt; &lt; CorrelationAuxInfo TaskId= {A61211FE-6D71- 4B90-B94B-16305E4A1944} SeqNumber= 14 /&gt;
&lt; Result value= 0 /&gt; &lt; /CertVerifyRevocation&gt; />

```

Aus dem Provider Microsoft-Windows-Kernel-File (26, 29), dem FileIO/Create-Event (27) des Windows Kernels sowie dem Microsoft-Windows-FileInfoMinifilter Provider (28) geht hervor, dass die heruntergeladene Ressource in das INetCache-Verzeichnis zwischengespeichert wird, bevor sie in das Ausführungsverzeichnis des DemoProcess unter dem Dateinamen favicon.png (30, 31) erstellt wird:

```

26 <Event MSec= "22391,3274" PID= "416" PName="DemoProcess" TID="7580" ActivityID="901e6d33591f0001166
f1f901f59d601" EventName="Create" ProviderName="Microsoft-Windows-Kernel-File" Irp="0xffffa90
3dc342a88" FileObject="0xffffa903d85af680" IssuingThreadId="7.580" CreateOptions="33.554.528"
CreateAttributes="8.192" ShareAccess="7" FileName="\Device\HarddiskVolume2\Users\JohnDoe\
AppData\Local\Microsoft\Windows\INetCache\IE\PQCBLA44\favicon_hsw_fiw[1].png"/>
27 <Event MSec= "22391,3291" PID= "416" PName="DemoProcess" TID="7580" EventName="FileIO/Create"
IrpPtr="0xFFFFA903DC342A88" FileObject="0xFFFFA903D85AF680" CreateOptions="FILE_ATTRIBUTE_
ARCHIVE, FILE_ATTRIBUTE_DEVICE" CreateDisposition="CREATE_ALWAYS" FileAttributes="NotContent
Indexed" ShareAccess="ReadWrite, Delete" FileName="C:\Users\JohnDoe\AppData\Local\Microsoft\
Windows\INetCache\IE\ PQCBLA44\favicon_hsw_fiw[1].png"/>
28 <Event MSec= "22392,0893" PID= "416" PName="DemoProcess" TID="7580" EventName="fi:FileNameCreate"
ProviderName="Microsoft-Windows-FileInfoMinifilter" FileObject="0xffffba0852391170"
Path="\Device\HarddiskVolume2\Users\JohnDoe\AppData\Local\Microsoft\Windows\INetCache\
IE\PQCBLA44\favicon_hsw_fiw[1].png"/>
29 <Event MSec= "22392,0914" PID= "416" PName="DemoProcess" TID="7580" EventName="NameCreate"
ProviderName="Microsoft-Windows-Kernel-File" FileKey="0xffffba0852391170"
FileName="\Device\HarddiskVolume2\Users\JohnDoe\AppData\Local\Microsoft\Windows\INetCache\IE\
PQCBLA44\favicon_hsw_fiw[1].png"/>
30 <Event MSec= "22403,2214" PID= "416" PName="DemoProcess" TID="4056" ActivityID="901e6d33591f000
15f6f1f901f59d601" EventName="Create" ProviderName="Microsoft-Windows-Kernel-File"
Irp="0xffffa903db9aba88" FileObject="0xffffa903d85af9a0" IssuingThreadId="4.056"
CreateOptions="83.886.176" CreateAttributes="128" ShareAccess="0"
FileName="\Device\HarddiskVolume2\etw\TV2favicon.png"/>

```



```

31 <Event MSec= "22403,5058" PID= "416" PName="DemoProcess" TID="4056" EventName="fi:FileNameCreate"
    ProviderName="Microsoft-Windows-FileInfoMinifilter" FileObject="0xffffba08527b6170"
    Path="\Device\HarddiskVolume2\etw\TV2favicon.png"/>

```

TV3: Lesen- und Schreiben und einer Datei

Für das Testverhalten 3 konnten die Events über das Prozessverhalten aus den Providern:

- Microsoft-Windows-FileInfoMinifilter
- Microsoft-Windows-Kernel-File
- Windows Kernel (Keywords: FileIO)

bezogen werden, um das Lesen- und Schreiben der Testdatei TV3CreateFileTest File.txt zu beobachten. Auszugsweise werden die Events (32-38) hervorgehoben:

```

32 <Event MSec= "28435,8014" PID= "416" PName="DemoProcess" TID="4056" ActivityID="901e6d33591f000fb
    0020901f59d601" EventName="Create" ProviderName="Microsoft-Windows-Kernel-File"
    Irp="0xfffffa903dc602788" FileObject="0xfffffa903d85b12a0" IssuingThreadId="4.056"
    CreateOptions="33.554.528" CreateAttributes="128" ShareAccess="1"
    FileName="\Device\HarddiskVolume2\etw\TV3CreateFileTestFile.txt"/>
33 <Event MSec= "28435,8033" PID= "416" PName="DemoProcess" TID="4056" EventName="FileIO/Create"
    IrpPtr="0xFFFFFA903DC602788" FileObject="0xFFFFFA903D85B12A0"
    CreateOptions="FILE_ATTRIBUTE_ARCHIVE, FILE_ATTRIBUTE_DEVICE" CreateDisposition="CREATE_ALWAYS"
    FileAttributes="Normal" ShareAccess="Read" FileName="C:\etw\TV3CreateFileTestFile.txt"/>
34 <Event MSec= "28436,3003" PID= "416" PName="DemoProcess" TID="4056"
    ActivityID="901e6d33591f000fb0020901f59d601" EventName="CreateNewFile" ProviderName=
    "Microsoft-Windows-Kernel-File" Irp="0xfffffa903dc602788" FileObject="0xfffffa903d85b12a0"
    IssuingThreadId="4.056" CreateOptions="33.554.528" CreateAttributes="128" ShareAccess="1"
    FileName="\Device\HarddiskVolume2\etw\TV3CreateFileTestFile.txt"/>
35 <Event MSec= "28436,3098" PID= "416" PName="DemoProcess" TID="4056" EventName="fi:FileNameCreate"
    ProviderName="Microsoft-Windows-FileInfoMinifilter" FileObject="0xffffba08544a6800"
    Path="\Device\HarddiskVolume2\etw\TV3CreateFileTestFile.txt"/>
36 <Event MSec= "28436,3117" PID= "416" PName="DemoProcess" TID="4056" EventName="NameCreate"
    ProviderName="Microsoft-Windows-Kernel-File" FileKey="0xffffba08544a6800"
    FileName="\Device\HarddiskVolume2\etw\TV3CreateFileTestFile.txt"/>
37 <Event MSec= "28436,9784" PID= "416" PName="DemoProcess" TID="4056" EventName="FileIO/Write"
    FileName="C:\etw\TV3CreateFileTestFile.txt" Offset="0" IrpPtr="0xFFFFFA903DC602788"
    FileObject="0xFFFFFA903D85B12A0" FileKey="0xFFFFFA08544A6800" IoSize="16" IoFlags="395.776"/>
38 <Event MSec= "33466,4649" PID= "416" PName="DemoProcess" TID="4056" EventName="FileIO/Read"
    FileName="C:\etw\TV3CreateFileTestFile.txt" Offset="0" IrpPtr="0xFFFFFA903DBD459E8"
    FileObject="0xFFFFFA903DB116C60" FileKey="0xFFFFFA08544A6800" IoSize="4.191" IoFlags="395.520"/>

```

TV4: Erstellung eines Kindprozesses

Für das Testverhalten 4 konnten die Events über das Prozessverhalten aus den Providern:

- Microsoft-Windows-FileInfoMinifilter
- Microsoft-Windows-Kernel-File
- Microsoft-Windows-Kernel-Registry
- Microsoft-Windows-Kernel-Prefetch
- Microsoft-Windows-Kernel-Process
- Windows Kernel (Keywords: FileIO, Image, Process, Registry)

bezogen werden.

Zunächst erfolgt eine Registry-Abfrage nach dem Executable zu WINVER (39, 40) und ein Dateizugriff (41). Das `Process/Start` (42) stellt Informationen zum Prozessstart des Prozesses `winver` mit dem Image `winver.exe` (43) dar. Es wird ein Handle zur Prefetch-Datei von `winver` erstellt (44) und die Datei ausgelesen (45). Im Provider `Microsoft-Windows-Kernel-Prefetch` werden über das Event `ScenarioDecision` (46) Prefetch-Informationen, wie die Anzahl der bisherigen Ausführungen und die vergangene Zeit seit der letzten Ausführung, bereitgestellt:

```

39 <Event MSec= "39518,2261" PID= "416" PName="DemoProcess" TID="4056" EventName="EventID(2)"
    ProviderName="Microsoft-Windows-Kernel-Registry" BaseObject="0xffffba08578e3770" KeyObject=
    "0x00000000" Status="-1.073.741.772" Disposition="0" BaseName="" RelativeName="winver.exe"/>
40 <Event MSec= "39518,2301" PID= "416" PName="DemoProcess" TID="4056" EventName="Registry/Open"
    Status="0x00000000" KeyHandle="0xFFFFBA084BC3A130" ElapsedTimeMSec="0,0302999999985332"
    KeyName="winver.exe" Index="0"/>
41 <Event MSec= "39518,3152" PID= "416" PName="DemoProcess" TID="4056"
    ActivityID="901e6d33591f000c45220901f59d601" EventName="Create" ProviderName="Microsoft-
    Windows-Kernel-File" Irp="0xffffa903dc602788" FileObject="0xffffa903d85b2ba0"
    IssuingThreadId="4.056" CreateOptions="16.777.312" CreateAttributes="128" ShareAccess="5"
    FileName="\Device\HarddiskVolume2\Windows\System32\winver.exe"/>
42 <Event MSec= "39521,2326" PID="5464" PName= "winver" TID= "-1" EventName="Process/Start"
    ProcessID="5.464" ParentID="416" ImageFileName="winver.exe" DirectoryTableBase="0xD17DB000"
    Flags="None" SessionID="1" ExitStatus="0x00000103" UniqueProcessKey="0xFFFFA903DC181080"
    CommandLine=" C:\Windows\System32\winver.exe "/>
43 <Event MSec= "39532,3019" PID="5464" PName= "winver" TID="4688" EventName="Image/Load" ImageBase=
    "0x00007FF66FD50000" ImageSize="0x00013000" ImageChecksum="83.271" TimeDateStamp=
    "1.414.741.537" DefaultBase="0x00007FF66FD50000" FileName="C:\Windows\System32\winver.exe"/>
44 <Event MSec= "39532,5205" PID="5464" PName= "winver" TID="4688" EventName="FileIO/Create"
    IrpPtr="0xFFFFA903DBD459E8" FileObject="0xFFFFA903D85B0620" CreateOptions="FILE_ATTRIBUTE_
    ARCHIVE" CreateDisposition="CREATE_NEW" FileAttributes="0" ShareAccess="None"
    FileName="C:\Windows\Prefetch\WINVER.EXE-D053C8CF.pf"/>
45 <Event MSec= "39533,1400" PID="5464" PName= "winver" TID="4688" EventName="FileIO/Read"
    FileName="C:\Windows\Prefetch\WINVER.EXE-D053C8CF.pf" Offset="0" IrpPtr="0xFFFFA903DC602788"
    FileObject="0xFFFFA903D85B0620" FileKey="0xFFFFBA0857FD9170" IoSize="8.192" IoFlags="394.243"/>
46 <Event MSec= "39552,6758" PID="5464" PName= "winver" TID="4688" EventName="ScenarioDecision"
    ProviderName="Microsoft-Windows-Kernel-Prefetch" ScenarioName="WINVER.EXE"
    ScenarioHashId="-799.815.473" ScenarioType="0" ActionFlags="3" TraceReason="13"
    PrefetchReason="13" NumLaunches="1" TimeSinceLastLaunchInS="630.329"/>

```

6.1.6 Bewertung der Ergebnisse

Aus den gewonnenen Ergebnissen nach Auswertung der prozessbezogenen Events aus den ETW-Providern zeigt sich, dass sich die Testverhaltensweisen TV1-TV4 in den aufgezeichneten Events vollständig zuordnen und nachweisen ließen. Aus 48 Providern (inkl. des Kernel Traces) wurden 50.398 Events aufgezeichnet.

Inhaltlich lassen sich den Eventdaten relevante Informationen wie Pfadangaben zu Registry-Schlüsseln zu Dateien entnehmen und in einen zeitlichen Zusammenhang bringen. Hierbei beinhalteten die aufgezeichneten Events keine Daten von (übertragenen) Dateien oder gesetzten bzw. ausgelesenen Registry-Werten. Geschriebene Werte oder Dateiinhalte ließen sich aus den einzelnen Events nicht ermitteln.

Teilweise konnten redundante Informationen zum Verhalten über Dateisystem- und Registryoperationen aus den Providern Microsoft-Windows-Kernel-File und Microsoft-Windows-Kernel-Registry auch aus dem Windows Kernel Providern (Keywords: FileIO und Registry) bezogen werden. Hierbei unterscheidet sich neben dem Informationsumfang auch die Darstellung von Pfaden.

In Anbetracht einer hohen Anzahl aufgezeichneter Events, die keine forensisch relevanten Informationen liefern und vielmehr zur Diagnose, Debugging oder Performancemessung ausgerichtet sind, gestaltet sich die Auswertung insgesamt komplex. Eine gezielte Filterung nach bestimmten Anhaltspunkten, wie beispielsweise Dateiendungen (.txt/.exe) oder Registry-Pfaden aus dem definierten Testverhalten oder bekannten Verhaltensweisen der MITRE Techniken kann hierbei unterstützen. Durch die Spaltenstruktur der exportierten CSV-Datei lassen Events nach Provider- und Event-Namen filtern.

6.2 Konkreter Anwendungsfall: njRAT (PoC II)

In PoC II soll eine Malware untersucht werden, die idealerweise bereits über alternative Analysemethoden untersucht worden ist. Hierzu eignet sich beispielsweise eine Malware, zu der es einen Analysebericht gibt, um später die aus ETW ermittelten Informationen gegenüberzustellen.

Bei njRAT handelt es sich um einen Remote Access Trojaner (RAT) mit dem ein Fernzugriff auf den infizierten Zielrechner realisiert wird. Häufig werden RATs über einen Dropper in das entsprechende Zielsystem des Opfers eingeschleust. Im Malware Analysis Blog von „Anurag“ wird eine njRAT Malware Analyse vorgestellt und analysiert [Anu20]. Der C2-Server beantwortet TCP-Anfragen nicht, sodass sich

die vorliegende Analyse auf den Infektionsweg (Dropper) fokussiert, anstatt dem Verhalten der njRAT-Komponente selbst. Über den Dienst „ANY RUN - Interactive Malware Analysis“ wird ein Analysebericht mit Aufzeichnung der Ausführung innerhalb einer Analyse-VM bereitgestellt [ANY20a]. Die Analyse im Rahmen dieser Arbeit wird sich an den beiden genannten Berichten orientieren.

Die Plattform ANY.RUN [ANY20a] ordnet das analysierte Malware-Sample nach MITRE ATT&CK Matrix wie folgt ein:

1. **Execution:**

- T1106 Execution through API

2. **Persistence:**

- T1060 Registry Run Keys / Startup Folder

3. **Defense Evasion:**

- T1089 Disabling Security Tools

6.2.1 Vorbereitung und Fokus der Analyse

Für die Analyse im PoC II wird das Malware-Sample (Listing 6.27) von der Plattform ANY.RUN auf das Analysesystem (entsprechend 6.1.1) heruntergeladen:

```
Malware-Sample: 123123.exe
MD5: 88e085572a182ca102676676ec0ef802
SHA1: 82ff7c328ccb0a57dc3d65f97b2f38b3b9127324
```

Listing 6.27: PoC II - Informationen zum njRAT Malware-Sample

Der Fokus der Analyse liegt in der Erkennung der in den Analyseberichten aufgeführten Verhaltensweisen, um die forensisch relevanten ETW-Provider bestimmen und zuordnen zu können.

6.2.2 Methodik der Aufzeichnung

Mit dem ETWProcessTracer wird eine Echtzeitaufzeichnung der ETW-Analysesitzung vorgenommen. Alle prozessbezogenen Events werden aus dem Datenstrom herausgefiltert und nach dem Parsing in eine CSV-Datei (Protokolldatei) exportiert. Die Aufzeichnung soll bis zur Terminierung der erzeugten Prozesse ausgeführt werden. Sollten nicht alle Prozesse terminieren, wird die Aufzeichnung manuell durch Benutzerinteraktion beendet.

6.2.3 Identifikation und Auswahl geeigneter ETW-Provider

Da die Anzahl der Provider zur Laufzeit des Prozesses dynamisch ist, werden alle auf dem Analysesystem zur Verfügung stehenden Provider in die Aufzeichnung einbezogen. Nach Abschluss der Analyse können forensisch relevante Provider bestimmt werden, um die aufgezeichneten Events dem jeweiligen Prozessverhalten zuzuordnen.

6.2.4 Aufzeichnen des Prozessverhaltens

Zur Aufzeichnung des Prozessverhaltens zum Malware-Sample (`123123.exe`) wird eine neue Eingabeaufforderung (`cmd.exe`) gestartet und die zugewiesene PID ermittelt. Diese Methode stellt eine alternative Vorgehensweise zur Ausführung mit Hilfe des `ProcessStartHelper` aus PoC I dar und kann beispielsweise herangezogen werden, wenn die Ausführung mit dem `ProcessStartHelper` fehlschlägt oder nicht gewünscht ist. Malware kann über verschiedene Wege zur Ausführung gebracht werden. Die Ausführung über die Kommandozeile (Eingabeaufforderung oder der PowerShell) sollte einer Ausführung über den klassischen Start per „Doppelklick“ jedoch vorgezogen werden, um mögliche Ausgaben auf der Kommandozeile beobachten zu können (siehe Listing 6.28):

```
c:\>tasklist | findstr -i cmd
cmd.exe                5108 Console                1        4.148 K
```

Listing 6.28: PoC II - Ermittlung der PID der Eingabeaufforderung (`cmd.exe`)

Die ermittelte PID der `CMD`, über die das Malware-Sample zur Ausführung gebracht werden soll, wird anschließend als Parameter an den `ETWProcessTracer` übergeben (Listing 6.29):

```
C:\> ETWProcessTracer.exe --PID 5108 --KernelKeywords All
[...]
CONFIGURATION:
-----
Target ProcessID:      5108
ProviderConfig:        all registered ETW-Providers
KernelKeywords:        All

Logfile directory:     C:\etw\ETWProcessTracer\
-----
[INFO] Press CTRL+C at any time to stop tracing
Press <Enter> to start monitoring...
```

Listing 6.29: PoC II - Start der Analyse mit dem `ETWProcessTracer`

Nachdem die ETW-Analysesitzung mit dem ETWProcessTracer gestartet wurde, werden prozessbezogene Ereignisse der PID 5108 (cmd.exe) aufgezeichnet. Hierzu zählen auch Ereignisse, die die Erstellung von Kindprozessen protokollieren. Die PIDs der durch den Prozess cmd (PID: 5108) Kindprozesse werden mit in die Prozess-Watchlist aufgenommen.

Das Malware-Sample 123123.exe wird über die mit erhöhten Rechten ausgeführte Eingabeaufforderung mit der PID 5108 zur Ausführung gebracht (Listing 6.30):

```
c:\>123123.exe
```

Listing 6.30: PoC II - Ausführung des njRAT Malware-Samples 123123.exe

Eine weitere Benutzerinteraktion ist nicht erforderlich. Ferner verfügt das Malware-Sample über keine Ausgaben auf der Konsole oder der GUI, die Informationen zur Ausführung exponieren. Aus Sicht des Benutzers entsteht der Eindruck, dass die vermeintliche Anwendung nicht funktioniert oder keine weitere Funktion hat.

Über die Konsolenausgabe des ETWProcessTracers lässt sich beobachten (siehe Listing 6.31), dass nach Ausführung des Malware-Samples 123123.exe eine hohe Anzahl von Events (ca. 37000 Events/s) erfasst werden und zusätzliche Kindprozesse erzeugt werden:

```
[PROCESS CREATED] PID: 8320 (PPID: 5108) ProcessName: "123123" ImagePath: "123123.exe" Args: "123123.exe"
[PROCESS CREATED] PID: 6660 (PPID: 8320) ProcessName: "svchost" ImagePath: "svchost.exe" Args: "C:\Users\JohnDoe\AppData\Roaming\svchost.exe"
[PROCESS TERMINATED] PID: 8320 (PPID: 5108) ProcessName: "" ImagePath: "123123.exe" Args: "123123.exe"
[PROCESS CREATED] PID: 5692 (PPID: 6660) ProcessName: "netsh" ImagePath: "netsh.exe" Args: "netsh firewall add allowedprogram "C:\Users\JohnDoe\AppData\Roaming\svchost.exe" "svchost.exe" ENABLE"
[PROCESS CREATED] PID: 1768 (PPID: 5692) ProcessName: "conhost" ImagePath: "conhost.exe" Args: "\??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1"
[PROCESS TERMINATED] PID: 5692 (PPID: 6660) ProcessName: "" ImagePath: "netsh.exe" Args: "netsh firewall add allowedprogram "C:\Users\JohnDoe\AppData\Roaming\svchost.exe" "svchost.exe" ENABLE"
[PROCESS TERMINATED] PID: 1768 (PPID: 5692) ProcessName: "" ImagePath: "conhost.exe" Args: "\??\C:\Windows\system32\conhost.exe 0x4"
```

Listing 6.31: PoC II - Konsolenausgabe des ETWProcessTracers zu njRAT

Der Kindprozess svchost (PID: 6660) terminiert auch nach längerer Beobachtung nicht. Dies stellt einen ersten Anhaltspunkt auf eine persistente Ausführung, z.B. eines Agents / Bots dar. Die Aufzeichnung wird daher nach Aufzeichnung von 1.999.822 Events durch Benutzereingabe STRG+C abgebrochen.

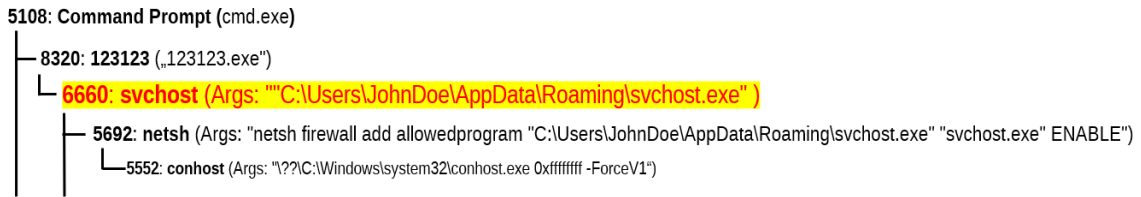


Bild 6.3: PoC II - Prozessbaum über die njRAT-Infektion

6.2.5 Auswertung der Aufzeichnung

In den aufgezeichneten Events der Provider

- Microsoft-Windows-Kernel-Process
- Windows Kernel (Keywords: Process, Image)
- Microsoft-Windows-Kernel-Prefetch

lassen sich die Ereignisse über den Prozessstart (1, 2) des Malware-Samples 123123.exe (PID: 8320) über die Eingabeaufforderung (cmd, PID: 5108) (1) entnehmen. Der Prozess 123123 wurde um 13:36:53 Uhr durch den Elternprozess (cmd, PID: 5108) erstellt, dabei wurde das Image C:\mw\123123.bin\123123.exe (3, 4) geladen. Der Wert -1 im Datenfeld NumLaunches des Events ScenarioDecision (5, 6) gibt an, dass die Datei 123123.exe erstmalig aus dem Verzeichnis C:\mw\123123.bin\ ausgeführt wird:

```

1 <Event MSec= "12681,9591" PID="5108" PName= "cmd" TID="6200" EventName="ProcessStart/Start"
  ProviderName="Microsoft-Windows-Kernel-Process" FormattedMessage="Process 8.320 started at
  time 13:36:53.088642 (12.681,705 MSec) by parent 5.108 running in session 1 with name &lt; &lt;
  BadFieldIdx&gt; &gt; . " ProcessID="8.320" ProcessSequenceNumber="343"
  CreateTime="13:36:53.088642 (12.681,705 MSec)" ParentProcessID="5.108"
  ParentProcessSequenceNumber="145" SessionID="1" Flags="0" ProcessTokenElevationType="2"
  ProcessTokenIsElevated="1"/>
2 <Event MSec= "12681,9669" PID="8320" PName= "123123" TID= "-1" EventName="Process/Start"
  ProcessID="8.320" ParentID="5.108" ImageFileName="123123.exe" DirectoryTableBase="0x836AB000"
  Flags="Wow64" SessionID="1" ExitStatus="0x0000103" UniqueProcessKey="0xFFFFFC988568C1080"
  CommandLine="123123.exe"/>
3 <Event MSec= "12701,2352" PID="8320" PName= "123123" TID="2224" EventName="Image/Load"
  ImageBase="0x00450000" ImageSize="0x0000E000" ImageChecksum="0" TimeDateStamp="1.592.316.264"
  DefaultBase="0x004000000000A000" FileName="C:\mw\123123.bin\123123.exe"/>
4 <Event MSec= "12701,2292" PID="8320" PName= "123123" TID="2224" EventName="ImageLoad"
  ProviderName="Microsoft-Windows-Kernel-Process" FormattedMessage="Process 8.320 had an image
  loaded with name \Device\HarddiskVolume2\mw\123123.bin\123123.exe. " ImageBase="0x00450000"
  ImageSize="0x0000e000" ProcessID="8.320" ImageCheckSum="0" TimeDateStamp="1.592.316.264"
  DefaultBase="0x4000000000a000" ImageName="\Device\HarddiskVolume2\mw\123123.bin\123123.exe"/>
5 <Event MSec= "12701,4169" PID="8320" PName= "123123" TID="2224"
  ActivityID="f97f6d3866650000ee0c80f96566d601" EventName="Create"
  ProviderName="Microsoft-Windows-Kernel-File" Irp="0xffffc988523640f8"
  FileObject="0xffffc98857177da0" IssuingThreadId="2.224" CreateOptions="16.777.248"
    
```

```

CreateAttributes="0" ShareAccess="0"
FileName="\Device\HarddiskVolume2\Windows\Prefetch\123123.EXE-6549D45E.pf"/>
6 <Event MSec= "12701,5253" PID="8320" PName= "123123" TID="2224" EventName="ScenarioDecision"
ProviderName="Microsoft-Windows-Kernel-Prefetch" ScenarioName="123123.EXE"
ScenarioHashId="1.699.337.310" ScenarioType="0" ActionFlags="1" TraceReason="13"
PrefetchReason="10" NumLaunches="-1" TimeSinceLastLaunchInS="-1"/>

```

Durch den Prozess 123123.exe (PID: 8320) werden zwei weitere Dateien (svchost.exe und Tools.exe) im Dateisystem erstellt („gedroppt“). Die zugehörigen Events Create (7), FileIO/Create (8), CreateNewFile (9) Write (10) und FileIO/Write (11, 12) lassen sich aus den Providern:

- Microsoft-Windows-Kernel-File
- Microsoft-Windows-FileInfoMinifilter
- Windows Kernel (Keywords: FileIO)

entnehmen. Auszugsweise werden hier exemplarisch die Events für die Datei Tools.exe sowie das Write-Event (13) zur Datei svchost.exe gelistet:

```

7 <Event MSec= "13269,9276" PID="8320" PName= "123123" TID="2224"
ActivityID="f97f6d3866650001e8fd7ff96566d601" EventName="Create"
ProviderName="Microsoft-Windows-Kernel-File" Irp="0xffffc9885256cc88"
FileObject="0xffffc9885717ba90" IssuingThreadId="2.224" CreateOptions="18.874.368"
CreateAttributes="0" ShareAccess="7" FileName="\Device\HarddiskVolume2\Tools.exe"/>
8 <Event MSec= "13269,9293" PID="8320" PName= "123123" TID="2224" EventName="FileIO/Create"
IrpPtr="0xFFFFC9885256CC88" FileObject="0xFFFFC9885717BA90" CreateOptions="2097152"
CreateDisposition="CREATE_NEW" FileAttributes="0" ShareAccess="ReadWrite, Delete"
FileName="C:\Tools.exe"/>
9 <Event MSec= "13272,2797" PID="8320" PName= "123123" TID="2224"
ActivityID="f97f6d3866650000da1d80f96566d601" EventName="CreateNewFile"
ProviderName="Microsoft-Windows-Kernel-File" Irp="0xffffc988523640f8"
FileObject="0xffffc9885717e330" IssuingThreadId="2.224" CreateOptions="33.554.500"
CreateAttributes="32" ShareAccess="0" FileName="\Device\HarddiskVolume2\Tools.exe"/>
10 <Event MSec= "13272,2886" PID="8320" PName= "123123" TID="2224" EventName="fi:FileNameCreate"
ProviderName="Microsoft-Windows-FileInfoMinifilter" FileObject="0xffffb38a02a17170"
Path="\Device\HarddiskVolume2\Tools.exe"/>
11 <Event MSec= "13274,6538" PID="8320" PName= "123123" TID="2224"
ActivityID="f97f6d3866650000f71d80f96566d601" EventName="Write"
ProviderName="Microsoft-Windows-Kernel-File" ByteOffset="0" Irp="0xffffc988523640f8"
FileObject="0xffffc9885717ba90" FileKey="0xffffb38a02a17170" IssuingThreadId="2.224"
IoSize="32.256" IoFlags="395.776" ExtraFlags="0"/>
12 <Event MSec= "13274,6551" PID="8320" PName= "123123" TID="2224" EventName="FileIO/Write"
FileName="C:\Tools.exe" Offset="0" IrpPtr="0xFFFFC988523640F8" FileObject="0xFFFFC9885717BA90"
FileKey="0xFFFFB38A02A17170" IoSize="32.256" IoFlags="395.776"/>
13 <Event MSec= "19392,3210" PID="8320" PName= "123123" TID="2224" EventName="FileIO/Write"
FileName="C:\Users\JohnDoe\AppData\Roaming\svchost.exe" Offset="0" IrpPtr="0xFFFFC98856EC2C88"
FileObject="0xFFFFC9885717E330" FileKey="0xFFFFB389FFF62170" IoSize="32.256"
IoFlags="395.776"/>

```

Für die Datei scvhost.exe werden diese Events analog erzeugt. Zusätzlich lässt sich für diese Datei beobachten, dass die Events:

- CDesktopFolder_ParseDisplayName/Start und Stop (14)
- ShellTask_ExecAssoc_ZoneCheckFile/Stop (15, 16)

des Providers Microsoft-Windows-Shell-Core aufgezeichnet wurden:

```

14 <Event MSec= "19401,2253" PID="8320" PName= "123123" TID="2224"
    EventName="CDesktopFolder_ParseDisplayName/Start" ProviderName="Microsoft-Windows-Shell-Core"
    Name="C:\Users\JohnDoe\AppData\Roaming\svchost.exe"/>
15 <Event MSec= "19551,6641" PID="8320" PName= "123123" TID="2224"
    EventName="CDesktopFolder_ParseDisplayName/Stop" ProviderName="Microsoft-Windows-Shell-Core"
    Name="C:\Users\JohnDoe\AppData\Roaming\svchost.exe" HRESULT="0" PIDL_out="9.639.104" HWND="0"
    IBindCtx="9.335.208" cbEaten="-1" dwAttributes="1.077.936.453"/>
16 <Event MSec= "19667,5480" PID="8320" PName= "123123" TID="2224"
    EventName="ShellTask_ExecAssoc_ZoneCheckFile/Stop" ProviderName="Microsoft-Windows-Shell-Core"
    psz="C:\Users\JohnDoe\AppData\Roaming\svchost.exe"/>

```

Einträge über die Datei `svchost.exe` werden in der Windows-Registry über die Provider:

- Microsoft-Windows-Kernel-Registry (17, 19, 22)
- Windows Kernel (Keywords: Registry) (18, 21)

abgerufen:

```

17 <Event MSec= "19753,1293" PID="8320" PName= "123123" TID="2224" EventName="EventID(2)"
    ProviderName="Microsoft-Windows-Kernel-Registry" BaseObject="0xffffb389f9e11940"
    KeyObject="0x00000000" Status="-1.073.741.772" Disposition="0" BaseName=""
    RelativeName="\Registry\Machine\Software\Microsoft\Windows
    NT\CurrentVersion\AppCompatFlags\Custom\svchost.exe"/>
18 <Event MSec= "19753,1314" PID="8320" PName= "123123" TID="2224" EventName="Registry/Open"
    Status="0x00000000" KeyHandle="0x00000000" ElapsedTimeMSec="0,0108999999974912"
    KeyName="\Registry\Machine\Software\Microsoft\Windows
    NT\CurrentVersion\AppCompatFlags\Custom\svchost.exe" Index="0"/>
19 <Event MSec= "19758,3440" PID="8320" PName= "123123" TID="2224" EventName="EventID(2)"
    ProviderName="Microsoft-Windows-Kernel-Registry" BaseObject="0xffffb38a046e9470"
    KeyObject="0x00000000" Status="-1.073.741.772" Disposition="0" BaseName=""
    RelativeName="Applications\svchost.exe"/>
20 <Event MSec= "19758,3464" PID="8320" PName= "123123" TID="2224" EventName="Registry/Open"
    Status="0x00000000" KeyHandle="0xFFFFFB389FE4C1410" ElapsedTimeMSec="0,0520999999971536"
    KeyName="Applications\svchost.exe" Index="0"/>
21 <Event MSec= "19758,3731" PID="8320" PName= "123123" TID="2224" EventName="Registry/Open"
    Status="0x00000000" KeyHandle="0x00000000" ElapsedTimeMSec="0,0165999999990163"
    KeyName="\Registry\Machine\Software\Classes\Applications\svchost.exe" Index="0"/>
22 <Event MSec= "19758,7716" PID="8320" PName= "123123" TID="2224" EventName="EventID(2)"
    ProviderName="Microsoft-Windows-Kernel-Registry" BaseObject="0xffffb38a046e9470"
    KeyObject="0x00000000" Status="-1.073.741.772" Disposition="0" BaseName=""
    RelativeName="Applications\svchost.exe"/>

```

Ein zusätzlicher Prozess `svchost.exe` (PID: 6660) Prozess wird gestartet (23) und das zugehörige Image aus dem `AppData`-Verzeichnispfad des Benutzers geladen (24):

```

23 <Event MSec= "19751,3990" PID="6660" PName= "svchost" TID= "-1" EventName="Process/Start"
    ProcessID="6.660" ParentID="8.320" ImageFileName="svchost.exe" DirectoryTableBase="0xBE0F1000"
    Flags="Wow64" SessionID="1" ExitStatus="0x00000103" UniqueProcessKey="0xFFFFC988523684C0"
    CommandLine=" C:\Users\JohnDoe\AppData\Roaming\svchost.exe "/>

```

```

24 <Event MSec= "19763,8533" PID="6660" PName= "svchost" TID="2636" EventName="Image/Load"
    ImageBase="0x00EC0000" ImageSize="0x0000E000" ImageChecksum="0" TimeDateStamp="1.592.316.264"
    DefaultBase="0x004000000000A000" FileName="C:\Users\JohnDoe\AppData\Roaming\svchost.exe"/>

```

Die in Windows integrierte „Network Shell (netsh)“ wird durch `svchost` (PID: 6660) dazu verwendet, um Netzwerkverkehr in der Windows-Firewall zu erlauben (siehe T1089). Anhand des Parameters `CommandLine` des `Process/Start`-Event (25) des Prozesses `netsh` (PID: 7252) kann entnommen werden, dass eine Firewallregel hinzugefügt werden soll. Das Hinzufügen der Firewallregel lässt sich außerdem aus dem Provider `Microsoft-Windows-MPS-CLNT` über das `Start/Stop`-Event `MPS_CLNT_API_SetFirewallRule` (26, 27) entnehmen. Die zugehörigen Events aus diesem Provider bestätigen jedoch lediglich, dass eine Firewallregel hinzugefügt wurde, geben jedoch keinen Aufschluss über Bezeichnung und Art der neu hinzugefügten Firewallregel. Einziger Anhaltspunkt stellt hier lediglich der Kommandoaufrufparameter der `Network Shell (netsh)` (25) dar:

```

25 <Event MSec= "28146,3164" PID="5692" PName= "netsh" TID= "-1" EventName="Process/Start"
    ProcessID="5.692" ParentID="6.660" ImageFileName="netsh.exe" DirectoryTableBase="0xBF317000"
    Flags="Wow64" SessionID="1" ExitStatus="0x00000103" UniqueProcessKey="0xFFFFFC988522E90C0"
    CommandLine="netsh firewall add allowedprogram C:\Users\JohnDoe\AppData\Roaming\svchost.exe
    svchost.exe ENABLE"/>
26 <Event MSec= "29184,7400" PID="5692" PName= "netsh" TID="7288"
    EventName="MPS_CLNT_API_SetFirewallRule/Start" ProviderName="Microsoft-Windows-MPS-CLNT"/>
27 <Event MSec= "29187,4808" PID="5692" PName= "netsh" TID="7288"
    EventName="MPS_CLNT_API_SetFirewallRule/Stop" ProviderName="Microsoft-Windows-MPS-CLNT"/>

```

Die ausführbare Datei `e84128b2e0547d1dd1f8090d86c80c48.exe` wird durch Hinterlegen in das `Startup`-Verzeichnis zu jeder Anmeldung des Benutzers automatisch gestartet (siehe T1060). Events über das Erstellen dieser Datei wurden über die Provider:

- Microsoft-Windows-Kernel-File
- Microsoft-Windows-FileInfoMinifilter
- Windows Kernel (Keywords: FileIO)

aufgezeichnet:

```

28 <Event MSec= "29320,2002" PID="6660" PName= "svchost" TID="2636" ActivityID="f97f6d38665000084b
    180f96566d601" EventName="Create" ProviderName="Microsoft-Windows-Kernel-File"
    Irp="0xffffc988523640f8" FileObject="0xffffc98856725780" IssuingThreadId="2.636"
    CreateOptions="83.886.148" CreateAttributes="32" ShareAccess="0"
    FileName="\Device\HarddiskVolume2\Users\JohnDoe\AppData\Roaming\Microsoft\ Windows\Start
    Menu\Programs\Startup\e84128b2e0547d1dd1f8090d86c80c48.exe"/>
29 <Event MSec= "29321,1117" PID="6660" PName= "svchost" TID="2636" EventName="NameCreate"
    ProviderName="Microsoft-Windows-Kernel-File" FileKey="0xffffb38a02d98700"
    FileName="\Device\HarddiskVolume2\Users\JohnDoe\AppData\Roaming\Microsoft\ Windows\Start
    Menu\Programs\Startup\e84128b2e0547d1dd1f8090d86c80c48.exe"/>

```

```
30 <Event MSec= "29323,5656" PID="6660" PName= "svchost" TID="2636" EventName="FileIO/Write"
    FileName="C:\Users\JohnDoe\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\
    e84128b2e0547d1dd1f8090d86c80c48.exe" Offset="0" IrpPtr="0xFFFFC98856E68838" FileObject=
    "0xFFFFC98856725780" FileKey="0xFFFFB38A02D98700" IoSize="32.256" IoFlags="395.776"/>
```

Zusätzlich werden Registry Run-Keys in der Registry hinterlegt, sodass das Executable `svchost.exe` bei jeder Anmeldung des Benutzers automatisch gestartet wird (siehe T1089). Zugehörige Events können aus den Providern:

- Microsoft-Windows-Kernel-Registry (33-36)
- Windows Kernel (Keywords: Registry) (31, 32)

entnommen werden:

```
31 <Event MSec= "29317,6951" PID="6660" PName= "svchost" TID="2636" EventName="Registry/KCBCreate"
    Status="0x00000000" KeyHandle="0xFFFFB38A01B24390" ElapsedTimeMSec="29317,6951" KeyName=
    "\REGISTRY\MACHINE\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Run" Index="0"/>
32 <Event MSec= "29317,8151" PID="6660" PName= "svchost" TID="2636"
    EventName="Registry/SetInformation" Status="0x00000000" KeyHandle="0xFFFFB38A01B24390"
    ElapsedTimeMSec="0,1064999999805" KeyName="" Index="0"/>
33 <Event MSec= "29317,7016" PID="6660" PName= "svchost" TID="2636" EventName="EventID(2)"
    ProviderName="Microsoft-Windows-Kernel-Registry" BaseObject="0xffffb389f9e11940"
    KeyObject="0xffffb38a0481bf70" Status="0" Disposition="0" BaseName=""
    RelativeName="\REGISTRY\MACHINE\Software\WOW6432Node\Microsoft\Windows\Current Version\Run"/>
34 <Event MSec= "30517,4185" PID="6660" PName= "svchost" TID="2636" EventName="EventID(7)"
    ProviderName="Microsoft-Windows-Kernel-Registry" KeyObject="0xffffb38a046f0570"
    Status="-1.073.741.772" InfoClass="2" DataSize="-1.761.406.132" KeyName=""
    ValueName="Software\Microsoft\Windows\CurrentVersion\Run\e84128b2e0547d1dd1f8090d86c80c48"
    CapturedData="" />
35 <Event MSec= "30522,4646" PID="6660" PName= "svchost" TID="2636" EventName="EventID(2)"
    ProviderName="Microsoft-Windows-Kernel-Registry" BaseObject="0xffffb389f9e11940"
    KeyObject="0xffffb38a0481e270" Status="0" Disposition="0" BaseName=""
    RelativeName="\REGISTRY\MACHINE\Software\WOW6432Node\Microsoft\Windows\Current Version\Run"/>
36 <Event MSec= "31544,4695" PID="6660" PName= "svchost" TID="2636" EventName="EventID(7)"
    ProviderName="Microsoft-Windows-Kernel-Registry" KeyObject="0xffffb38a046f0570"
    Status="-1.073.741.772" InfoClass="2" DataSize="23.243.304" KeyName=""
    ValueName="Software\Microsoft\Windows\CurrentVersion\Run\e84128b2e0547d1dd1f8090 d86c80c48"
    CapturedData="" />
```

Die eingetragenen Registry-Werte (Data) lassen sich aus dem Registry Editor entnehmen (siehe Bild 6.4). Aus den aufgezeichneten Events verbleibt im Datenfeld `CapturedData` (36) lediglich ein leerer String). Bild 6.4 zeigt einen Auszug der Windows-Registry, in der sich auch der geschriebene String entnehmen lässt:

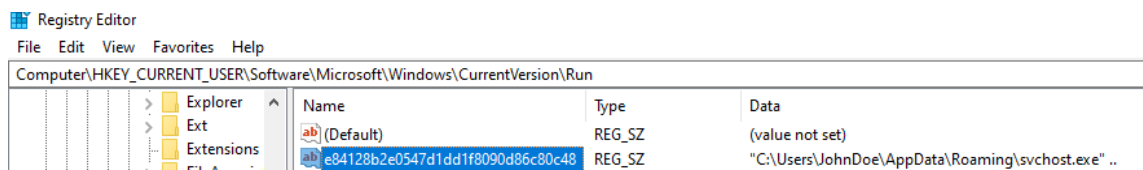


Bild 6.4: PoC II - njRat Registry-Run KeyValue

Aus den Providern Microsoft-Windows-Kernel-Process sowie dem Kernel Provider (Keyword: Image) geht hervor, dass zu einem späteren Zeitpunkt des eigentlichen Prozesses unter anderem Bibliotheken der Windows Sockets 2 eingebunden werden. Die `ws2_32.dll` und `mwssock.dll` der Windows Sockets 2 Bibliothek stellen einen Anhaltspunkt für eine Socket-Implementierung im Programmcode des Prozesses dar:

```

38 <Event MSec= "31804,9545" PID="6660" PName= "svchost" TID="1152" EventName="Image/Load"
    ImageBase="0x778A0000" ImageSize="0x0005E000" ImageChecksum="431.034"
    TimeDateStamp="48.530.558" DefaultBase="0x778A000000044000"
    FileName="C:\Windows\SysWOW64\ws2_32.dll"/>
39 <Event MSec= "31815,2444" PID="6660" PName= "svchost" TID="1152" EventName="Image/Load"
    ImageBase="0x70B60000" ImageSize="0x00052000" ImageChecksum="331.505"
    TimeDateStamp="-115.561.978" DefaultBase="0x70B6000000048000"
    FileName="C:\Windows\SysWOW64\mwssock.dll"/>

```

Der Netzwerkverkehr lässt sich aus Events der Provider:

- Microsoft-Windows-TCPIP
- Microsoft-Windows-Winsock-AFD
- Windows Kernel (Keyword: Tcpip)

entnehmen. Das Event `AfdConnectWithAddress/Bound` (40) zeigt, dass der Port 7777 des Zielhosts 85.26.235.163 durch den Prozess `svchost.exe` (PID: 6660) gebunden werden soll. Mit dem Event `TcpRequestConnect` (41) und `TcpConnectTcb Proceeding` (42) wurde der TCP-Verbindungsanfrage aufgezeichnet. Das aus dem Windows Kernel Trace erfasste `TcpIp/Reconnect`-Event (43) als auch das Event `TcpConnectRestrtransmit` (44) des Providers Microsoft-Windows-TCPIP weisen auf eine nicht zustande gekommene TCP-Verbindung hin. Auf das Event `TcpRequest Connect` (42) folgte nicht das zu erwartende Event `TcpBindEndpointComplete`, das eine erfolgreiche Bindung des Sockets bestätigt.

```

40 <Event MSec= "31927,4074" PID="6660" PName= "svchost" TID="1152" ActivityID="55eb32e0c988ffff000000
    0000000000" EventName="AfdConnectWithAddress/Bound"
    ProviderName="Microsoft-Windows-Winsock-AFD" FormattedMessage="connect: 0: Process
    0xffffc988523684c0, Endpoint 0xffffc98855eb32e0, Address 85.26.235.163 :7777, Seq 5.023,
    Status 0 " EnterExit="0" Location="5.023" Process="0xffffc988523684c0"
    Endpoint="0xffffc98855eb32e0" Buffer="0x00000000" BufferLength="0" Status="0"
    Address="85.26.235.163:7777"/>
41 <Event MSec= "31927,4580" PID="6660" PName= "svchost" TID="1152" ActivityID="505a19a0c988ffff000000
    0000000000" EventName="TcpRequestConnect" ProviderName="Microsoft-Windows-TCPIP"
    FormattedMessage="TCP: Tcb 0xffffc988505a19a0 (local=10.0.2.15:50040 remote=85.26.235.163:7777
    ) requested to connect. " Tcb="0xffffc988505a19a0" LocalAddress="10.0.2.15:50040"
    RemoteAddress="85.26.235.163:7777" NewState="ClosedState" RexmitCount="0"/>
42 <Event MSec= "31927,5118" PID="6660" PName= "svchost" TID="1152" ActivityID="505a19a0c988ffff000000
    0000000000" EventName="TcpConnectTcbProceeding" ProviderName= "Microsoft-Windows-TCPIP"
    FormattedMessage="TCP: connection 0xffffc988505a19a0 (local=10.0.2.15:50040 remote=
    85.26.235.163:7777) connect proceeding. " LocalAddress="10.0.2.15:50040" RemoteAddress=
    "85.26.235.163:7777" Status="0" ProcessId="0" Compartment="0" Tcb="0xffffc988505a19a0"/>

```

```

43 <Event MSec= "32929,2855" PID="6660" PName= "svchost" TID= "-1" EventName="TcpIp/Reconnect"
    daddr="85.26.235.163" saddr="10.0.2.15" dport="7.777" sport="50.040" size="0"
    connid="0x0F33001800000000" seqnum="255.000.600"/>
44 <Event MSec= "32929,2917" PID="6660" PName= "svchost" TID="3404" ActivityID="505a19a0c988ffff000000
    0000000000" EventName="TcpConnectRetransmit" ProviderName= "Microsoft-Windows-TCPIP"
    FormattedMessage="TCP: connection 0xffffc988505a19a0 (local=10.0.2.15:50040
    remote=85.26.235.163:7777) retransmitting connect attempt, RexmitCount = 1. "
    Tcb="0xffffc988505a19a0" LocalAddress="10.0.2.15:50040" RemoteAddress="85.26.235.163:7777"
    NewState="ClosedState" RexmitCount="1"/>

```

6.2.6 Bewertung der Ergebnisse

Nachfolgend aufgelistete ETW-Provider lieferten forensisch relevante Events zu den im Analysebericht der Plattform ANY.RUN angegebenen MITRE Techniken:

1. Execution (T1106)

- Microsoft-Windows-Kernel-Process
- Microsoft-Windows-Kernel-Prefetch
- Microsoft-Windows-Shell-Core
- Windows Kernel (Keyword: Process, ImageLoad)

2. Persistence (T1060)

- Microsoft-Windows-Kernel-File
- Microsoft-Windows-Kernel-Registry
- Windows Kernel (Keyword: FileIO, Registry)

3. Defense Evasion (T1089)

- Microsoft-Windows-MPS-CLNT

Bereits während der Aufzeichnung ließ sich als Zwischenergebnis auf der Konsolenausgabe erkennen, dass der initial ausgeführte Prozess `123123.exe` einen weiteren Prozess (`svchost`) erzeugt, der einen untypischen Image-Pfad (für den gleichnamigen Systemprozess des `Service Hosts`) zum AppData-Verzeichnis des Benutzers aufweist siehe T1106). Im weiteren Verlauf der Aufzeichnung wurde außerdem die Network Shell (`netsh`) aufgerufen, um für den `svchost`-Prozess eine Freigabe in der Firewall einzurichten. Die von ANY.RUN eingeordneten Techniken der MITRE ATTACK Matrix T1106 (Execution through API) und T1089 (Disabling Security Tools) ließen sich mit den in der Auswertung aufgelisteten Events zuordnen. Ebenfalls konnte aus den aufgezeichneten Events der Windows-Registry und dem Dateisystem entnehmen, dass die Malware persistent auf dem Analysesystem konfiguriert wird (siehe T1060).

Die Ergebnisse aus den aufgezeichneten Events aus den ETW-Providern, stimmen mit der statischen Code-Analyse von Anurag [Anu20] sowie der Live-Analyse der Plattform ANY.RUN überein [ANY20a].

Festzuhalten ist die sehr hohe Anzahl von aufgezeichneten Events. Über eine Zeitspanne von 1 Minute und 15 Sekunden wurden 1.999.822 Events aus 48 ETW-Providern (inkl. des Windows Kernel Traces) aufgezeichnet. Die hohe Anzahl aufgezeichneter Events, ohne forensisch relevante Informationen zum Prozessverhalten, erschweren die Auswertung. Eine Filterung der CSV-Protokolldatei nach den bereits ermittelten ETW-Providern mit forensisch relevanten Daten (aus PoC I) sowie nach MITRE Techniken und Informationen aus dem Analysebericht von Anurag [Anu20] unterstützt die Zuordnung der prozessspezifischen Verhaltensweisen.

6.3 Konkreter Anwendungsfall: Emotet (PoC III)

Im PoC III wird ein Microsoft Word-Dokument mit einem maliziösen Office-Makro analysiert, welches über den Windows Scripting Host (WSH) und die PowerShell den Trojaner „Emotet“ aus dem Internet nachlädt und zur Ausführung bringt. Maliziöse Word-Dokumente mit enthaltenen Makrovirus werden häufig über Spamwellen verbreitet. Im vorliegenden PoC sollen die Analyseergebnisse der Plattform ANY.RUN [ANY20b], des CERT Polska [Sro17] und virusbulletin [Nag19] zur Referenz herangezogen werden.

In der Plattform ANY.RUN werden die verwendeten Techniken des zu analysierenden Malware-Samples nach MITRE ATT&CK Matrix wie folgt zugeordnet [ANY20b]:

1. Execution / Defense Evasion

- T1106 Execution through API
- T1086 PowerShell
- T1064 Scripting
- T1047 Windows Management Instrumentation

2. Persistence

- T1060 Registry Run Keys / Startup Folder

3. Defense Evasion

- T1064 Scripting

4. Discovery

- T1012 Query Registry

5. Lateral Movement / Command & Control (C2)

- T1105 Remote File Copy

6.3.1 Vorbereitung und Fokus der Analyse

Für die Analyse im PoC III wird das Malware-Sample (Listing 6.32) von der Plattform ANY.RUN auf das Analysesystem heruntergeladen:

```
Malware-Sample: Invoice_503_292647.doc
MD5:      344ed3bad67d1f286c95237102047151
SHA1:     a2b4a2e3187f105f292812c326433cb267f6777f
```

Listing 6.32: PoC III - Informationen zum Emotet Malware-Sample

Dem Analysebericht der Plattform ANY.RUN ist zu entnehmen, dass die zu analysierende Malware die Windows Management Instrumentation (WMI) verwendet, um PowerShell-Skripte und Prozesse indirekt auszuführen [ANY20b]. Prozesse, die nicht vom initial überwachten Prozess (Parameter `--PID` im ETWProcessTracer) abstammen, werden nicht automatisch in die Analyse miteinbezogen. Dem ETWProcessTracer wird daher als Parameter `--SpawnDetection` der Prozessname `WmiPrvSE` übergeben. Die Erzeugung von Kindprozessen wird hierdurch sowohl für bereits ausgeführte Instanzen des WMI Provider Host Service (WmiPrvSE) als auch für neu gespawnte Prozessinstanzen des WmiPrvSE überwacht. Alle Kindprozesse werden in die Analyse mitaufgenommen und zugehörige Events aufgezeichnet.

6.3.2 Methodik der Aufzeichnung

Es wird eine Echtzeitaufzeichnung über die ETW-Analysesitzung des ETWProcessTracers (gem. Kapitel 6.1.4) vorgenommen. Dazu werden alle prozessbezogenen Ereignisse der initial angegebenen Prozess-ID von Microsoft Word (`WINWORD`) als auch erzeugten Kindprozessen der WmiPrvSE aus dem Datenstrom herausgefiltert, geparkt und in die CSV-Logdatei exportiert. Die Aufzeichnung soll bis zur Terminierung der (erzeugten) Prozesse ausgeführt werden. Sollten nicht alle beobachteten Prozesse terminieren, wird die Aufzeichnung manuell beendet.

6.3.3 Identifikation und Auswahl geeigneter ETW-Provider

Über den Parameter `--KernelKeywords All` werden alle verfügbaren Keywords der NT Kernel Logger Session aktiviert, sodass Events aus allen Kernel Providern

aufgezeichnet werden. Der Parameter `--ProviderConfig` wird nicht angegeben, sodass standardmäßig automatisch alle im System registrierten ETW-Provider durch den `ETWProcessTracer` mit in die Analyse einbezogen werden.

6.3.4 Aufzeichnen des Prozessverhaltens

Zur Aufzeichnung des Prozessverhaltens zum Malware-Sample (`Invoice_503_292647.doc`) wird zunächst Microsoft Word 2013 gestartet und die zugewiesene PID über die Kommandozeile ermittelt (Listing 6.33):

```
C:\>tasklist | findstr -i word
WINWORD.exe                7684 Console                1        58.148 K
```

Listing 6.33: PoC III - Ermittlung der PID der Eingabeaufforderung (`cmd.exe`)

Das zu untersuchende Word-Dokument mit dem Makro wird erst zu einem späteren Zeitpunkt geöffnet, um die beim Programmstart von Microsoft Word erzeugten ETW-Events nicht mit aufzuzeichnen zu müssen. Die ermittelte PID des `WINWORD`-Prozesses (PID: 7684), über den das Malware-Sample zur Ausführung gebracht werden soll, wird anschließend als Parameter an den `ETWProcessTracer` übergeben. Zusätzlich wird durch den Parameter `--SpawnDetection` die `WmiPrvSE` angegeben, sodass alle durch die `WmiPrvSE` erstellen Kindprozesse mit in die Watchlist des `ETWProcessTracers` aufgenommen werden (Listing 6.34):

```
C:\>ETWProcessTracer.exe --PID 7684 --KernelKeywords All --SpawnDetection WmiPrvSE
[...]
CONFIGURATION:
-----
Target ProcessID:      7684
SpawnDetection:       WmiPrvSE
ProviderConfig:       all registered ETW-Providers
KernelKeywords:       All

Logfile directory:    C:\etw\ETWProcessTracer\
-----
[INFO] Press CTRL+C at any time to stop tracing
Press <Enter> to start monitoring...
```

Listing 6.34: PoC III - Start der Analyse mit dem `ETWProcessTracer`

Nach Initiierung der ETW-Analysesitzung im `ETWProcessTracers`, wird das maliziöse Word-Dokument (`Invoice_503_292647.doc`) in Microsoft Word 2013 auf dem Analysesystem geöffnet (siehe Bild 6.5):

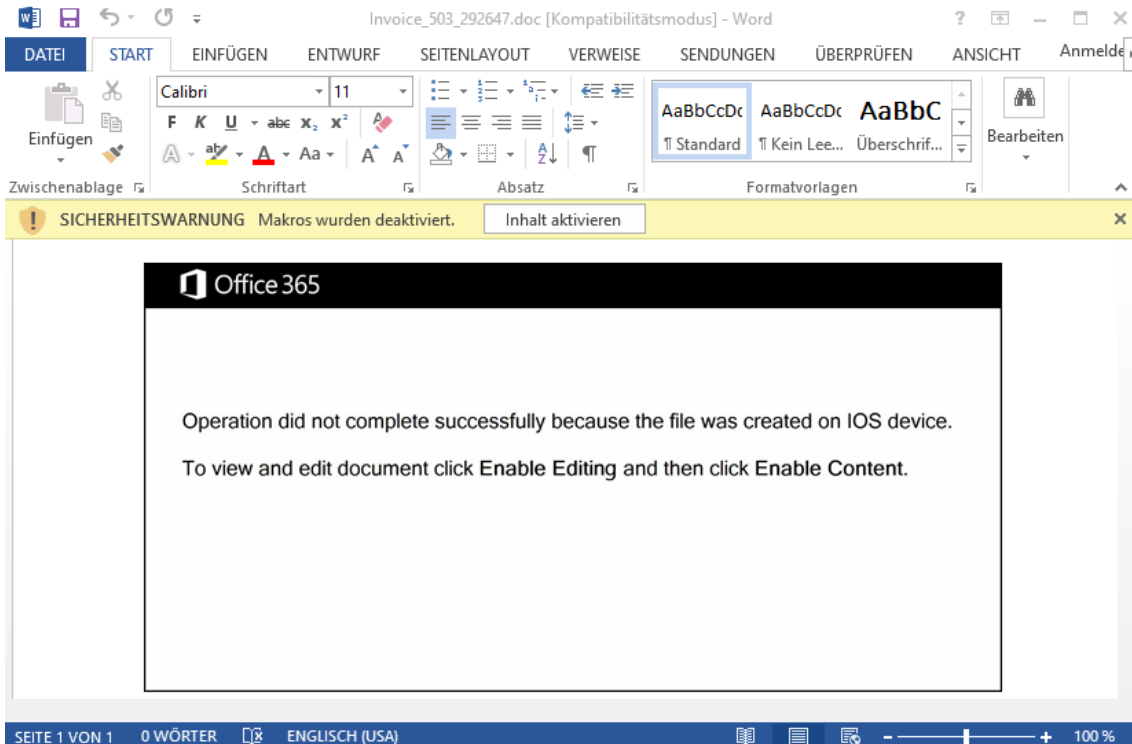


Bild 6.5: PoC III - Inhalt der Datei: Invoice_503_292647.doc

Damit das enthaltene **maliziöse** Makro tatsächlich zur Ausführung gebracht wird, ist durch Benutzerinteraktion die Schaltfläche „Inhalt aktivieren“ der Sicherheitswarnung in Microsoft Word zu betätigen.

Anschließend lässt sich auf der Konsolenausgabe des ETWProcessTracers (siehe Listing 6.35) beobachten:

(Anm.: Der obfuskierte PowerShell-Befehl wird zur besseren Lesbarkeit gekürzt dargestellt. Auf der Kommandozeile wird ein Teil des Parameters abgeschnitten.)

```
[SpawnDetection] Already running "WmiPrvSE" processes (SpawnDetection only):
    1008    WmiPrvSE
    7992    WmiPrvSE

[PROCESS CREATED] PID: 2520 (PPID: 7992) ProcessName: "powershell" ImagePath: "
    powershell.exe" Args: "powersheLL -e JABWAE[...]EEAdw"
[PROCESS CREATED] PID: 5552 (PPID: 2520) ProcessName: "conhost" ImagePath: "conhost
    .exe" Args: "\??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1"

[PROCESS CREATED] PID: 6168 (PPID: 7992) ProcessName: "751" ImagePath: "751.exe"
    Args: "C:\Users\JohnDoe\751.exe"

[PROCESS TERMINATED] PID: 2520 (PPID: 7992) ProcessName: "" ImagePath: "powershell.
    exe" Args: "powersheLL -e JABWAE[...]EEAdw"
[PROCESS TERMINATED] PID: 5552 (PPID: 2520) ProcessName: "" ImagePath: "conhost.exe
    " Args: "\??\C:\Windows\system32\conhost.exe 0x4"
```

```

[PROCESS CREATED] PID: 2008 (PPID: 6168) ProcessName: "FXSRESM" ImagePath: "FXSRESM
.exe" Args: "\"C:\Users\JohnDoe\AppData\Local\winml\FXSRESM.exe\""
[PROCESS TERMINATED] PID: 6168 (PPID: 7992) ProcessName: "" ImagePath: "751.exe"
  Args: "C:\Users\JohnDoe\751.exe"

[PROCESS CREATED] PID: 6488 (PPID: 2008) ProcessName: "FXSRESM" ImagePath: "FXSRESM
.exe" Args: "\"C:\Users\JohnDoe\AppData\Local\winml\FXSRESM.exe" "C:\Users\
JohnDoe\AppData\Local\Temp\3910.tmp\""
[PROCESS TERMINATED] PID: 6488 (PPID: 2008) ProcessName: "" ImagePath: "FXSRESM.exe"
  Args: "\"C:\Users\JohnDoe\AppData\Local\winml\FXSRESM.exe" "C:\Users\JohnDoe\
AppData\Local\Temp\3910.tmp\""

[PROCESS CREATED] PID: 4328 (PPID: 2008) ProcessName: "FXSRESMoe" ImagePath: "
FXSRESMoe.exe" Args: "\"C:\Users\JohnDoe\AppData\Local\winml\FXSRESMoe.exe" "C:\
Users\JohnDoe\AppData\Local\Temp\3910.tmp\""
[PROCESS TERMINATED] PID: 4328 (PPID: 2008) ProcessName: "" ImagePath: "FXSRESMoe.
exe" Args: "\"C:\Users\JohnDoe\AppData\Local\winml\FXSRESMoe.exe" "C:\Users\
JohnDoe\AppData\Local\Temp\3910.tmp\""

[PROCESS CREATED] PID: 2924 (PPID: 2008) ProcessName: "FXSRESM" ImagePath: "FXSRESM
.exe" Args: "\"C:\Users\JohnDoe\AppData\Local\winml\FXSRESM.exe" /scomma "C:\
Users\JohnDoe\AppData\Local\Temp\5B7D.tmp\""
[PROCESS TERMINATED] PID: 2924 (PPID: 2008) ProcessName: "" ImagePath: "FXSRESM.exe"
  Args: "\"C:\Users\JohnDoe\AppData\Local\winml\FXSRESM.exe" /scomma "C:\Users\
JohnDoe\AppData\Local\Temp\5B7D.tmp\""

[PROCESS CREATED] PID: 6816 (PPID: 2008) ProcessName: "FXSRESM" ImagePath: "FXSRESM
.exe" Args: "\"C:\Users\JohnDoe\AppData\Local\winml\FXSRESM.exe" /scomma "C:\
Users\JohnDoe\AppData\Local\Temp\63DB.tmp\""
[PROCESS TERMINATED] PID: 6816 (PPID: 2008) ProcessName: "" ImagePath: "FXSRESM.exe"
  Args: "\"C:\Users\JohnDoe\AppData\Local\winml\FXSRESM.exe" /scomma "C:\Users\
JohnDoe\AppData\Local\Temp\63DB.tmp\""

```

Listing 6.35: PoC III - Konsolenausgabe des ETWProcessTracers zu Emotet

6.3.5 Auswertung der Aufzeichnung

Nach Abschluss der Aufzeichnung wird die durch den ETWProcessTracer erzeugte Logdatei im CSV-Format ausgewertet. Die Events zum Prozessstart/-ende der erzeugten Prozesse und der geladenen Images lässt sich den Providern:

- Microsoft-Windows-Kernel-Process
- Windows Kernel (Keyword: Process, Image)

entnehmen. Hierbei ist zu erkennen, dass durch den WmiPrvSE-Prozess (PID: 7992) ein PowerShell-Prozess mit der PID 2520 erzeugt wird. Der Prozess stammt nicht von der initial überwachten PID 7684 (WINWORD.EXE) ab, da das ausgeführte Makro über die Windows Management Instrumentation (WMI) den neuen Prozess erstellt hat (z.B. mittels `CreateObject.Shell` in VBScript). Das geladene Image

zur PowerShell stammt aus dem System32-Verzeichnis, die dem ImageLoad-Event zu entnehmen ist. Zur Prozesserzeugung wird mit Parameter `-e` (EncodedCommand) ein Base64-kodierter PowerShell-Befehlsaufruf übergeben, der durch die PowerShell zur Ausführung gebracht werden soll.

In Bild 6.6 wird der Prozessbaum über die Emotet-Infektion auf dem Analysesystem grafisch dargestellt:

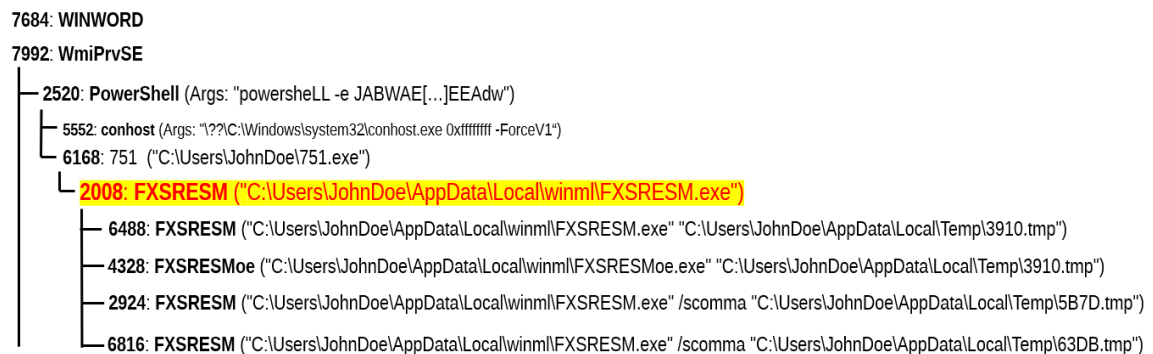


Bild 6.6: PoC III - Prozessbaum über die Emotet-Infektion

Die aufgezeichneten Image/Load-Events (2) WINWORD.EXE-Prozess (1) geben Aufschluss darüber, dass die WMI durch den WINWORD-Prozess verwendet wird. Die geladenen Images (`wbemdisp.dll`, `wbemcomn.dll`, `wbemprox.dll`, `wmiutils.dll`, `wbemsvc.dll`, `fastprox.dll`) sind typisch für den Aufruf eines WMI Befehls z.B. über ein Makro innerhalb eines Word-Dokuments [Fre19]:

```

1 <Event MSec= "0,0000" PID="7684" PName= "WINWORD" TID= "-1" EventName="Process/DCStart"
  ProcessID="7.684" ParentID="4.144" ImageFileName="WINWORD.EXE" DirectoryTableBase="0xC7152000"
  Flags="Wow64" SessionID="1" ExitStatus="0x00000103" UniqueProcessKey="0xFFFFFBFB85AE04C0"
  CommandLine=" C:\Program Files\Microsoft Office 15\Root\Office15\WINWORD.EXE /n
  C:\Users\JohnDoe\Desktop\Invoice_503_292647.doc /o "/>
2 <Event MSec= "4280,0933" PID="7684" PName= "WINWORD" TID="7840" EventName="ImageLoad"
  ProviderName="Microsoft-Windows-Kernel-Process" FormattedMessage="Process 7.684 had an image
  loaded with name \Device\HarddiskVolume2\Windows\SysWOW64\wbem\wmiutils.dll. "
  ImageBase="0x65e70000" ImageSize="0x0001d000" ProcessID="7.684" ImageChecksum="117.447"
  TimeDateStamp="2.028.077.747" DefaultBase="0x65e7000000017000"
  ImageName="\Device\HarddiskVolume2\Windows\SysWOW64\wbem\wmiutils.dll"/>

```

Der Base64 kodierte PowerShell Befehlsaufruf wird nicht vollständig im Event `Process/Start` (3) dargestellt. Im Datenfeld `CommandLine` werden max. 512 Zeichen dargestellt. Dem Event `StartingProvider/Tobeusedwhenoperationisjustexecutingmethod` (5) des Providers `Microsoft-Windows-PowerShell` kann jedoch der vollständige obfuskierte Befehl entnommen werden. (Anm.: *Der PowerShell-Befehlsaufruf wird nur auszugsweise gelistet*):

```

3 <Event MSec= "4700,9231" PID="2520" PName="powershell" TID= "-1" EventName="Process/Start"
  ProcessID="2.520" ParentID="7.992" ImageFileName="powershell.exe"
  DirectoryTableBase="0xD5599000" Flags="None" SessionID="1" ExitStatus="0x00000103"
  UniqueProcessKey="0xFFFFFBF0B819EC080" CommandLine="powershell -e -e JABWAE0AQ[...]BOAEEAdw"/>
4 <Event MSec= "4703,2062" PID="2520" PName="powershell" TID="5568" EventName="Image/Load"
  ImageBase="0x00007FF7E8700000" ImageSize="0x00071000" ImageChecksum="497.254"
  TimeDateStamp="-52.005.875" DefaultBase="0x00007FF7E8700000"
  FileName="C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe"/>
5 <Event MSec= "8205,2842" PID="2520" PName="powershell" TID="3968"
  ActivityID="1f3d28c269a70001e2a3841fa769d601"
  EventName="StartingProvider/Tobeusedwhenoperationisjustexecutingamethod"
  ProviderName="Microsoft-Windows-PowerShell" FormattedMessage="Provider Registry changed state
  to Started. Context: Severity = Informational Host Name = ConsoleHost Host Version =
  5.1.18362.752 Host ID = 0e14299e-11e0-4e2d-9aef-a2e35042f913 Host Application = powershell -e
  JABWA[M...]AJWA= Engine Version = Runspace ID = Pipeline ID = Command Name =
  Command Type = Script Name = Command Path = Sequence Number = 2
  User = DESKTOP-01E0000\JohnDoe Connected User = Shell ID = Microsoft.PowerShell User
  Data: ContextInfo= Severity = Informational Host Name = ConsoleHost Host Version
  = 5.1.18362.752 Host ID = 0e14299e-11e0-4e2d-9aef-a2e35042f913 Host Application =
  powershell -e JABWA[M...]AJWA= Engine Version = Runspace ID = Pipeline ID =
  Command Name = Command Type = Script Name = Command Path =
  Sequence Number = 2 User = DESKTOP-01E0000\JohnDoe Connected User = Shell ID =
  Microsoft.PowerShell UserData= Payload=Provider Registry changed state to Started. "/>

```

Der obfuskierte Skriptcode bzw. die vollständigen verwendeten Befehlsaufrufe lassen sich aus den Datenfeldern der Events nicht entnehmen. Die verwendete Cmdlets, Methoden und Skript-Aufrufe werden über die Events:

- StartingProvider/Tobeusedwhenoperationisjustexecutingamethod
- StartingCommand/Tobeusedwhenoperationisjustexecutingamethod
- StoppingEngine/Tobeusedwhenoperationisjustexecutingamethod

im Provider Microsoft-Windows-PowerShell bereitgestellt.

Aus weiteren aufgezeichneten Events in der Protokolldatei (nicht gelistet) geht hervor, dass folgende Befehle ausgeführt wurden:

- Script execution (Start)
- Out-Default
- Import-Module
- New-Object
- Get-Item
- out-lineoutput
- format-default
- Script execution (Stop)



Informationen zum obfuskieren PowerShell-Befehlsaufruf

Um die Funktionsaufrufe des PowerShell-EncodedCommand für die weitere Auswertung unterstützend einbeziehen zu können, wird der aus den Events entnehmbare obfuskierter PowerShell-Befehlsaufruf über die Funktion `[System.Text.Encoding]::Unicode.GetString([System.Convert]::FromBase64String)` dekodiert und ausgewertet. Bereits vollständig dekodierte PowerShell-Befehlsaufrufe ließen sich den aufgezeichneten Events nicht entnehmen.

Über den obfuskieren PowerShell-Befehlsaufruf werden folgende Funktionen ausgeführt:

1. Erlauben von TLS 1.0, 1.1 und 1.2 für Webanfragen aus der PowerShell (über `[Net.ServicePointManager]::SecurityProtocol`).
2. Anlegen einer Variable die auf den Pfad zum Benutzerprofilverzeichnis und den Dateinamen `751.exe` verweist.
3. Erstellen eines `Net.WebClient`-Objekts.
4. Anlegen eines Arrays mit 3 URLs.
5. Herunterladen einer Datei über die Funktion `DownloadFile` des WebClients in das Verzeichnis des Benutzers (Die 1. URL wird verwendet, bei dem die heruntergeladene Datei eine Dateigröße von 29604 Bytes oder größer aufweist).
6. Ausführen der heruntergeladenen Datei über die Funktion `Create` der WMI-Klasse `win_32_Process`.

Aus dem Provider `Microsoft-Windows-DotNETRuntime` wird durch das Event `Exception/Start` (6) für den Prozess der PowerShell (PID: 2520) erfasst, dass die Werte `tls12`, `tls11` und `tls` nicht vorgefunden wurden. Außerdem offenbart das Event `Method/InliningFailed` (7) aufgrund eines fehlgeschlagenen Inlinings, dass die im Inline Namespace `System.Net.WebClient` enthaltene `DownloadFile`-Funktion verwendet wurde:

```
6 <Event MSec= "11418,8532" PID="2520" PName="powershell" TID="1312"
  ActivityID="1f3d28c269a7000dad08d1fa769d601" EventName="Exception/Start"
  ProviderName="Microsoft-Windows-DotNETRuntime"
  FormattedMessage="ExceptionType=System.ArgumentException ExceptionMessage=Requested value
  &apos; tls12, tls11, tls&apos; was not found. ExceptionEIP=0x7ffeb46bedad
  ExceptionHRESULT=-2.147.024.809 ExceptionFlags=CLSCompliant ClrInstanceID=9 "
  ExceptionType="System.ArgumentException" ExceptionMessage="Requested value &apos; tls12, tls11,
  tls&apos; was not found." ExceptionEIP="0x7ffeb46bedad" ExceptionHRESULT="-2.147.024.809"
  ExceptionFlags="CLSCompliant" ClrInstanceID="9"/>
7 <Event MSec= "14996,0239" PID="2520" PName="powershell" TID="1312"
  ActivityID="1f3d28c269a7000194ba841fa769d601" EventName="Method/InliningFailed"
  ProviderName="Microsoft-Windows-DotNETRuntime"
  FormattedMessage="MethodBeingCompiledNamespace=dynamicClass
  MethodBeingCompiledName=CallSite.Target MethodBeingCompiledNameSignature=class System.Object
```

```
(pMT: 00007FFEB15ADE20,pMT: 00007FFEB15ACE80,class System.Object,class System.Object,class
System.Object) InlinerNamespace=dynamicClass InlinerName=CallSite.Target
InlinerNameSignature=class System.Object (pMT: 00007FFEB15ADE20,pMT: 00007FFEB15ACE80,class
System.Object,class System.Object,class System.Object) InlineeNamespace=System.Net.WebClient
InlineeName=DownloadFile InlineeNameSignature=instance void (class System.String,class
System.String) FailAlways=False FailReason=requires same this ClrInstanceID=9 "
MethodBeingCompiledNamespace="dynamicClass" MethodBeingCompiledName="CallSite.Target"
MethodBeingCompiledNameSignature="class System.Object (pMT: 00007FFEB15ADE20,pMT:
00007FFEB15ACE80,class System.Object,class System.Object,class System.Object)"
InlinerNamespace="dynamicClass" InlinerName="CallSite.Target" InlinerNameSignature="class
System.Object (pMT: 00007FFEB15ADE20,pMT: 00007FFEB15ACE80,class System.Object,class
System.Object,class System.Object)" InlineeNamespace="System.Net.WebClient"
InlineeName="DownloadFile" InlineeNameSignature="instance void (class System.String,class
System.String)" FailAlways="False" FailReason="requires same this" ClrInstanceID="9"/>
```

Die Datei 751.exe wird durch den PowerShell-Prozess (PID: 2520) im Benutzerverzeichnis angelegt (8, 9):

```
8 <Event MSec= "15009,2631" PID="2520" PName="powershell" TID="1312" EventName="FileIO/Create"
  IrpPtr="0xFFFFF0B82EB20F8" FileObject="0xFFFFF0B85C1E8B0" CreateOptions="4194400"
  CreateDisposition="TRUNCATE_EXISTING" FileAttributes="0" ShareAccess="Read"
  FileName="C:\Users\JohnDoe\751.exe"/>
9 <Event MSec= "15009,5349" PID="2520" PName="powershell" TID="1312"
  ActivityID="1f3d28c269a70001febd841fa769d601" EventName="CreateNewFile"
  ProviderName="Microsoft-Windows-Kernel-File" Irp="0xffffbf0b82eb20f8"
  FileObject="0xffffbf0b85c1e8b0" IssuingThreadId="1.312" CreateOptions="88.080.480"
  CreateAttributes="0" ShareAccess="1" FileName="\Device\HarddiskVolume2\Users\JohnDoe\751.exe"/>
```

Die PowerShell (PID: 2520) bindet die Windows DNS-API Bibliothek ein (10) und richtet einen RPC-Aufruf an den DNSResolver (11), um eine DNS-Namensauflösung für die im PowerShell-Skript enthaltene URL durchzuführen:

```
10 <Event MSec= "15135,1929" PID="2520" PName="powershell" TID="1312" EventName="ImageLoad"
  ProviderName="Microsoft-Windows-Kernel-Process" FormattedMessage="Process 2.520 had an image
  loaded with name \Device\HarddiskVolume2\Windows\System32\dnsapi.dll. "
  ImageBase="0x7ffed7c50000" ImageSize="0x000cb000" ProcessID="2.520" ImageChecksum="825.605"
  TimeDateStamp="-1.246.115.726" DefaultBase="0x7ffed7c50000"
  ImageName="\Device\HarddiskVolume2\Windows\System32\dnsapi.dll"/>
11 <Event MSec= "15189,4394" PID="2520" PName="powershell" TID="1312"
  ActivityID="3a4bce1877034b45af27d9afa05a8ad5" EventName="RpcClientCall/Start"
  ProviderName="Microsoft-Windows-RPC" FormattedMessage="Client RPC call started.
  InterfaceUuid: 45776b01-5956-4485-9f80-f428f7d60129 OpNum: 4 Protocol: LRPC
  NetworkAddress NULL Endpoint DNSResolver Binding Options NULL Authentication Level 7
  Authentication Service 8 Impersonation Level 9 "
  InterfaceUuid="45776b01-5956-4485-9f80-f428f7d60129" ProcNum="4" Protocol="LRPC"
  NetworkAddress="NULL" Endpoint="DNSResolver" Options="NULL" AuthenticationLevel="Packet
  Privacy" AuthenticationService="Kernel" ImpersonationLevel="Impersonate"/>
```

Aus den Providern Microsoft-Windows-DNS-Client und Microsoft-Windows-Winsock-NameResolution lässt sich das Ergebnis der DNS-Namensauflösungen über die Events mit der ID 3006 (12), 3008 (13) sowie WinsockGai (14) ermitteln. Zur Domain prolicitar[.]com[.]br wurde die IP-Adresse 177.52.160.72 aufgelöst. Events

über zusätzliche Namensauflösungen der im PowerShell enthaltenen weiteren URLs wurden nicht aufgezeichnet:

```

12 <Event MSec= "15183,6257" PID="2520" PName="powershell" TID="1312"
    ActivityID="1f3d28c269a7000194ba841fa769d601" EventName="EventID(3006)"
    ProviderName="Microsoft-Windows-DNS-Client" FormattedMessage="DNS query is called for the name
    prolicitar.com.br, type 28, query options 1.073.897.472, Server List , isNetwork query 0,
    network index 0, interface index 0, is asynchronous query 0 " QueryName="prolicitar.com.br"
    QueryType="28" QueryOptions="1.073.897.472" ServerList="" IsNetworkQuery="0"
    NetworkQueryIndex="0" InterfaceIndex="0" IsAsyncQuery="0"/>
13 <Event MSec= "15419,4601" PID="2520" PName="powershell" TID="1312"
    ActivityID="1f3d28c269a7000194ba841fa769d601" EventName="EventID(3008)"
    ProviderName="Microsoft-Windows-DNS-Client" FormattedMessage="DNS query is completed for the
    name prolicitar.com.br, type 28, query options 2.251.800.887.582.720 with status 0 Results
    ::ffff:177.52.160.72 " QueryName="prolicitar.com.br" QueryType="28"
    QueryOptions="2.251.800.887.582.720" QueryStatus="0" QueryResults="::ffff:177.52.160.72 "/>
14 <Event MSec= "15419,4970" PID="2520" PName="powershell" TID="1312"
    ActivityID="1f3d28c269a7000194ba841fa769d601" EventName="WinsockGai"
    ProviderName="Microsoft-Windows-Winsock-NameResolution" FormattedMessage="NSPLookupServiceNext
    is completed for provider 22059d40-7e9e-11cf-ae5a-00aa00a7112b, control Flags 0 and lookup
    Handle 2.209.106.181.456 with status 0 and result [::ffff:177.52.160.72]:53 "
    ProviderGUID="22059d40-7e9e-11cf-ae5a-00aa00a7112b" ControlFlags="0"
    LookupHandle="2.209.106.181.456" Status="0" Result="[:ffff:177.52.160.72]:53 "/>

```

Eine TCP-Verbindung wird zum Endpunkt 177.52.160.72 auf den Ziel-Port 80 über einen Socket durch die PowerShell hergestellt, wie aus den Events (15-16) der Provider Microsoft-Windows-TCPIP und Microsoft-Windows-Winsock-AFD hervorgeht. In den Events des Windows Kernel Traces wird durch die Tcpip/Recv- und Tcpip/Recv-Events (17-19) sichtbar, dass über eine Dauer von 1,65 s Daten von Host 177.52.160.72 empfangen wurden:

```

15 <Event MSec= "15448,6653" PID="2520" PName="powershell" TID="1312"
    ActivityID="847e57e0bf0bffff0000000000000000" EventName="AfdConnectWithAddress/Bound"
    ProviderName="Microsoft-Windows-Winsock-AFD" FormattedMessage="connect: 0: Process
    0xffffbf0b819ec080, Endpoint 0xffffbf0b847e57e0, Address 177.52.160.72:80, Seq 5.023, Status 0
    " EnterExit="0" Location="5.023" Process="0xffffbf0b819ec080" Endpoint="0xffffbf0b847e57e0"
    Buffer="0x00000000" BufferLength="0" Status="0" Address="177.52.160.72:80"/>
16 <Event MSec= "15448,7048" PID="2520" PName="powershell" TID="1312"
    ActivityID="85aed320bf0bffff0000000000000000" EventName="TcpRequestConnect"
    ProviderName="Microsoft-Windows-TCPIP" FormattedMessage="TCP: Tcb 0xffffbf0b85aed320
    (local=10.0.2.15:49923 remote=177.52.160.72:80) requested to connect. "
    Tcb="0xffffbf0b85aed320" LocalAddress="10.0.2.15:49923" RemoteAddress="177.52.160.72:80"
    NewState="ClosedState" RexmitCount="0"/>
17 <Event MSec= "16014,6177" PID="2520" PName="powershell" TID= "-1" EventName="TcpIp/Recv"
    daddr="177.52.160.72" saddr="10.0.2.15" dport="80" sport="49.923" size="2.880"
    connid="0x00007FFE00000000" seqnum="32.766"/>
18 <Event MSec= "17668,4997" PID="2520" PName="powershell" TID= "-1" EventName="TcpIp/Recv"
    daddr="177.52.160.72" saddr="10.0.2.15" dport="80" sport="49.923" size="1.140"
    connid="0x397E93AE00000000" seqnum="964.596.654"/>
19 <Event MSec= "21154,4067" PID="2520" PName="powershell" TID= "-1" EventName="TcpIp/Disconnect"
    daddr="177.52.160.72" saddr="10.0.2.15" dport="80" sport="49.923" size="0"
    connid="0xFFFFFBF0B0000000" seqnum="-16.629"/>

```


Aus parallel zu den TcpIp/Recv-Events (17-19) aufgezeichneten FileIO/Write-Events (20, 21) aus dem Windows Kernel Trace lässt sich korrelieren, dass Daten in die zuvor angelegte Datei 751.exe geschrieben werden:

```

20 <Event MSec= "16028,7922" PID="2520" PName="powershell" TID="1312" EventName="FileIO/Write"
    FileName="C:\Users\JohnDoe\751.exe" Offset="0" IrpPtr="0xFFFFBFB0B81DA0598"
    FileObject="0xFFFFBFB0B85C1E8B0" FileKey="0xFFFFE70262EFB170" IoSize="8.046" IoFlags="395.776"/>
21 <Event MSec= "17668,9130" PID="2520" PName="powershell" TID="1312" EventName="FileIO/Write"
    FileName="C:\Users\JohnDoe\751.exe" Offset="942.678" IrpPtr="0xFFFFBFB0B85D8AD28"
    FileObject="0xFFFFBFB0B85C1E8B0" FileKey="0xFFFFE70262EFB170" IoSize="2.986" IoFlags="0"/>

```

Der PowerShell-Prozess (PID: 2520) bindet Bibliotheken der Windows Management Instrumentation (WMI) in den Prozess ein:

- wmiutils.dll
- wbemcomn.dll
- wbemprox.dll
- wbemsvc.dll
- fastprox.dll

Zusätzlich wird die Bibliothek WMINet_Utils.dll geladen, wie dem Image/Load-Event aus dem Windows Kernel Trace oder alternativ dem Provider Microsoft-Windows-Kernel-Process zu entnehmen ist (auszugsweise werden hier lediglich die Events zur WMINet_Utils.dll aus dem Image/Load-Event (22) des Windows Kernel Traces aufgelistet):

```

22 <Event MSec= "19804,4089" PID="2520" PName="powershell" TID="4900" EventName= "Image/Load"
    ImageBase="0x00007FFECA660000" ImageSize="0x00030000" ImageChecksum="238.892"
    TimeDateStamp="1.551.504.840" DefaultBase="0x00007FFECA660000"
    FileName="C:\Windows\Microsoft.NET\Framework64\v4.0.30319\WMINet_Utils.dll" />

```

Der Prozess 751 (PID: 6168) wird mit dem, über die PowerShell, heruntergeladenen Image (751.exe) im Benutzerverzeichnis gestartet (23):

```

23 <Event MSec= "20126,3684" PID="6168" PName= "751" TID= "-1" EventName="Process/Start"
    ProcessID="6.168" ParentID="7.992" ImageFileName="751.exe" DirectoryTableBase="0x5D21D000"
    Flags="Wow64" SessionID="1" ExitStatus="0x00000103" UniqueProcessKey="0xFFFFBFB0B857D1080"
    CommandLine="C:\Users\JohnDoe\751.exe"/>

```

Aus dem RenamePath-Event (24) des Microsoft-Windows-Kernel-File Providers sowie des Shell32_CopyEngine_FileOperation_Info-Events (25) des Providers Microsoft-Windows-Shell-Core geht hervor, dass durch Prozess 751 das Image 751.exe in ein neues Verzeichnis kopiert wird.

```

24 <Event MSec= "21155,4484" PID="6168" PName= "751" TID="7924"
    ActivityID="1f3d28c269a700011cfb841fa769d601" EventName="RenamePath"
    ProviderName="Microsoft-Windows-Kernel-File" Irp="0xfffffbf0b847a5d28"
    FileObject="0xfffffbf0b85c1e590" FileKey="0xffffe70262efb170" ExtraInformation="0x00000000"

```



```

    IssuingThreadId="7.924" InfoClass="10"
    FilePath="\Device\HarddiskVolume2\Users\JohnDoe\AppData\Local\winml\FXSRESM.exe" />
25 <Event MSec= "21160,8674" PID="6168" PName= "751" TID="7924"
    EventName="Shell32_CopyEngine_FileOperation_Info" ProviderName="Microsoft-Windows-Shell-Core"
    pszSource="C:\Users\JohnDoe\751.exe"
    pszDest="C:\Users\JohnDoe\AppData\Local\winml\FXSRESM.exe" SourceType="3" DestinationType="3"
    FileOp="1" FileSize="945.664"/>

```

Der Kindprozess FXSRESM 2008 wird durch den Prozess 751 (6168) erzeugt (siehe Process/Start-Event, 26, 27). Anschließend terminiert Prozess 751 (6168) (28).

```

26 <Event MSec= "21309,5156" PID="2008" PName= "FXSRESM" TID= "-1" EventName="Process/Start"
    ProcessID="2.008" ParentID="6.168" ImageFileName="FXSRESM.exe" DirectoryTableBase="0x459C3000"
    Flags="Wow64" SessionID="1" ExitStatus="0x00000103" UniqueProcessKey="0xFFFFFBFB85AEF080"
    CommandLine=" C:\Users\JohnDoe\AppData\Local\winml\FXSRESM.exe" />
27 <Event MSec= "21309,5114" PID="6168" PName= "751" TID="7924" EventName="ProcessStart/Start"
    ProviderName="Microsoft-Windows-Kernel-Process" FormattedMessage="Process 2.008 started at
    time 17:24:22.175642 (21.309,189 MSec) by parent 6.168 running in session 1 with name &lt; &lt;
    BadFieldIdx&gt; &gt; . " ProcessID="2.008" ProcessSequenceNumber="317"
    CreateTime="17:24:22.175642 (21.309,189 MSec)" ParentProcessID="6.168"
    ParentProcessSequenceNumber="316" SessionID="1" Flags="0" ProcessTokenElevationType="3"
    ProcessTokenIsElevated="0"/>
28 <Event MSec= "21327,3553" PID="6168" PName= "" TID= "-1" EventName="Process/Stop"
    ProcessID="6.168" ParentID="7.992" ImageFileName="751.exe" DirectoryTableBase="0x5D21D000"
    Flags="Wow64" SessionID="1" ExitStatus="0x00000000" UniqueProcessKey="0xFFFFFBFB857D1080"
    CommandLine="C:\Users\JohnDoe\751.exe"/>

```

Aus dem Windows Kernel Trace (Keyword: Registry) und dem Provider Microsoft-Windows-Kernel-Registry lässt sich entnehmen (29, 30, 34), dass der Prozess FXSRESM (2008) den aktuellen Computernamen und die kryptografische MachineGUID, als eindeutigen Identifier des Clientsystems, ausliest. Darüber hinaus ist in den Image/Load-Events (31, 32) des Kernel Traces zu erkennen, dass weitere Images des Windows Task Managers (`taskmgr.exe`) und des HTML-Anwendungshosts (`mshta.exe`) in den Prozess geladen werden. Zu den geladenen Images zählen auch Bibliotheken der Windows Sockets (`mwssock.dll`) und der Win32 Internet-Funktionen (`WinInet.dll`):

```

29 <Event MSec= "21394,8826" PID="2008" PName= "FXSRESM" TID="1788" EventName="Registry/Open"
    Status="0x00000000" KeyHandle="0x00000000" ElapsedTimeMSec="0,0156000000024505"
    KeyName="\Registry\Machine\System\CurrentControlSet\Control\ComputerName\ ActiveComputerName"
    Index="0"/>
30 <Event MSec= "21394,9023" PID="2008" PName= "FXSRESM" TID="1788" EventName="EventID(7)"
    ProviderName="Microsoft-Windows-Kernel-Registry" KeyObject="0xfffffe70269946570" Status="0"
    InfoClass="1" DataSize="80" KeyName="" ValueName="ComputerName" CapturedData="" />
31 <Event MSec= "21391,1907" PID="2008" PName= "FXSRESM" TID="1788" EventName="Image/Load"
    ImageBase="0x001F0000" ImageSize="0x00008000" ImageChecksum="54.016"
    TimeDateStamp="823.174.628" DefaultBase="0x001F000000003000"
    FileName="C:\Windows\SysWOW64\mshta.exe"/>
32 <Event MSec= "21407,2253" PID="2008" PName= "FXSRESM" TID="1788" EventName="Image/Load"
    ImageBase="0x022D0000" ImageSize="0x000EA000" ImageChecksum="952.440"
    TimeDateStamp="1.719.021.756" DefaultBase="0x022D00000000BE000"
    FileName="C:\Windows\SysWOW64\Taskmgr.exe"/>

```

```

33 <Event MSec= "22586,3064" PID="2008" PName= "FXSRESM" TID="1788" EventName="Registry/Open"
    Status="0x00000000" KeyHandle="0xFFFFFE7025C641C20" ElapsedTimeMSec="0,01450000001135"
    KeyName="Software\Microsoft\Cryptography" Index="0"/>
34 <Event MSec= "22586,4762" PID="2008" PName= "FXSRESM" TID="1788" EventName="EventID(7)"
    ProviderName="Microsoft-Windows-Kernel-Registry" KeyObject="0xfffffe70268c05d70"
    Status="-2.147.483.643" InfoClass="2" DataSize="86" KeyName="" ValueName="MachineGuid"
    CapturedData=""/>

```

Durch den Prozess FXSRESM mit der PID 2008 wird ein HTTP POST-Request zum Zielobjekt 6ZKMSced/ an den Webserver 187.64.128.197 auf Port 80 erstellt (35) und übermittelt (36, 37). Zusätzliche Informationen werden über einen optionalen Header übertragen, wie aus Event WININET_REQUEST_HEADER_OPTIONAL (38) hervorgeht. Das Event WININET_RESPONSE_HEADERWININET_RESPONSE_HEADER (39) beinhaltet die HTTP-Response Header des nginx-Webrowsers mit der IP-Adresse 187.64.128.197:

```

35 <Event MSec= "35001,6556" PID="2008" PName= "FXSRESM" TID="1788"
    ActivityID="00cc000c00060000d807fc06102b8902" EventName="WININET_HTTP_REQUEST_HANDLE_CREATED"
    ProviderName="Microsoft-Windows-WinINet" FormattedMessage="Request handle 0x00cc000c created
    by HttpOpenRequest: ConnectionHandle=0x00cc0008, POST, Target=6ZKMSced/, Ver=HTTP/1.1,
    Referrer=, Media types=, Flags=-2.075.344.128 " ConnectionHandle="0x00cc000c"
    ParentHandle="0x00cc0008" Verb="POST" ObjectName="6ZKMSced/" Version="HTTP/1.1" Referrer=""
    AcceptTypes="" Flags="-2.075.344.128"/>
36 <Event MSec= "35001,7588" PID="2008" PName= "FXSRESM" TID="1788"
    ActivityID="00cc000c00060000d807fc06102b8902" EventName="Wininet_SendRequest/Start"
    ProviderName="Microsoft-Windows-WinINet" Request="0x02892b10"
    AddressName="http://187.64.128.197/6ZKMSced/">
37 <Event MSec= "35359,2639" PID="2008" PName= "FXSRESM" TID="1848"
    ActivityID="00cc000c00060000d807fc06102b8902" EventName="WININET_REQUEST_HEADER"
    ProviderName="Microsoft-Windows-WinINet" FormattedMessage="HTTP Request Headers
    RequestHandle=0x00cc000c POST /6ZKMSced/ HTTP/1.1 Referer: http://187.64.128.197/6ZKMSced/
    Content-Type: multipart/form-data boundary=-----742816567139757
    User-Agent: Mozilla/4.0 (compatible MSIE 7.0 Windows NT 6.2 WOW64 Trident/7.0 .NET4.0C
    .NET4.OE .NET CLR 2.0.50727 .NET CLR 3.0.30729 .NET CLR 3.5.30729) Host: 187.64.128.197
    Content-Length: 4756 Connection: Keep-Alive Cache-Control: no-cache "
    RequestHandle="0x00cc000c" Headers="POST /6ZKMSced/ HTTP/1.1 Referer:
    http://187.64.128.197/6ZKMSced/ Content-Type: multipart/form-data
    boundary=-----742816567139757 User-Agent: Mozilla/4.0 (compatible MSIE
    7.0 Windows NT 6.2 WOW64 Trident/7.0 .NET4.0C .NET4.OE .NET CLR 2.0.50727 .NET CLR 3.0.30729
    .NET CLR 3.5.30729) Host: 187.64.128.197 Content-Length: 4756 Connection: Keep-Alive
    Cache-Control: no-cache "/>
38 <Event MSec= "35360,2262" PID="2008" PName= "FXSRESM" TID="1848"
    ActivityID="00cc000c00060000d807fc06102b8902" EventName="WININET_REQUEST_HEADER_OPTIONAL"
    ProviderName="Microsoft-Windows-WinINet" FormattedMessage="HTTP Request Headers OptionalData
    RequestHandle=0x00cc000c -----742816567139757 Content-Disposition:
    form-data name= BQuJWDgFD filename= buhyrWAXwXkeXyRdZUP Content-Type: application/octet-stream
    [Anm.: nicht darstellbar] "/>
39 <Event MSec= "36791,8497" PID="2008" PName= "FXSRESM" TID="1848"
    ActivityID="00cc000c00060000d807fc06102b8902" EventName="WININET_RESPONSE_HEADER"
    ProviderName="Microsoft-Windows-WinINet" FormattedMessage="HTTP Response Headers
    RequestHandle=0x00cc000c HTTP/1.1 200 OK Server: nginx Date: Mon, 03 Aug 2020 13:24:47 GMT
    Content-Type: text/html charset=UTF-8 Content-Length: 144884 Connection: keep-alive Vary:
    Accept-Encoding " RequestHandle="0x00cc000c" Headers="HTTP/1.1 200 OK Server: nginx Date: Mon,

```

```
03 Aug 2020 13:24:47 GMT Content-Type: text/html charset=UTF-8 Content-Length: 144884
Connection: keep-alive Vary: Accept-Encoding "/>
```

Die Inhalte der empfangenen Antworten (Response-Body) des Webservers lassen sich dem Event WININET_HTTP_RESPONSE_BODY_RECEIVED (40) des Providers Microsoft-Windows-WinINet entnehmen. Der aufgezeichnete Response-Payload in den Events mit dem Namen EventID(2004) (41-43) des Providers Microsoft-Windows-WinINet-Capture beinhalten den aufgezeichneten Payload. Bei den hexadezimal dargestellten Daten handelt es sich allerdings nicht um ASCII-kodierten Text, der menschenlesbar konvertiert werden könnte:

```
40 <Event MSec= "39766,7980" PID="2008" PName= "FXSRESM" TID="1848"
    ActivityID="00cc000c00060000d807fc06102b8902" EventName="WININET_HTTP_RESPONSE_BODY_RECEIVED"
    ProviderName="Microsoft-Windows-WinINet" FormattedMessage="HTTP response entity body received:
    RequestHandle=0x00cc000c " RequestHandle="0x00cc000c"/>
41 <Event MSec= "39766,9595" PID="2008" PName= "FXSRESM" TID="1848"
    ActivityID="00cc000c00060000d807fc06102b8902" EventName="EventID(2004)"
    ProviderName="Microsoft-Windows-WinINet-Capture" FormattedMessage="The WinINet response
    payload buffer captured " SessionId="1" SequenceNumber="0" Flags="1"
    Payload="F7B6941A94791605043102A3116A6857...">
42 <Event MSec= "39767,0458" PID="2008" PName= "FXSRESM" TID="1848"
    ActivityID="00cc000c00060000d807fc06102b8902" EventName="EventID(2004)"
    ProviderName="Microsoft-Windows-WinINet-Capture" FormattedMessage="The WinINet response
    payload buffer captured " SessionId="1" SequenceNumber="1" Flags="0"
    Payload="72C0646459195FE0DA47DC3AD0632018...">
43 <Event MSec= "39767,0631" PID="2008" PName= "FXSRESM" TID="1848"
    ActivityID="00cc000c00060000d807fc06102b8902" EventName="EventID(2004)"
    ProviderName="Microsoft-Windows-WinINet-Capture" FormattedMessage="The WinINet response
    payload buffer captured " SessionId="1" SequenceNumber="2" Flags="0"
    Payload="9DAFBFA73200084D3AD0B16D2C8FCD54...">
```

Aus dem Event Wininet_UsageLogRequest (44) wird der abgewickelte HTTP POST-Request des HTTP Headers zusammengefasst. Die Funktion zum Übersenden von Formulardaten wird dazu verwendet, um mit dem C2-Server zu kommunizieren:

```
44 <Event MSec= "39767,1744" PID="2008" PName= "FXSRESM" TID="1788"
    ActivityID="00cc000c00060000d807fc06102b8902" EventName="Wininet_UsageLogRequest"
    ProviderName="Microsoft-Windows-WinINet" FormattedMessage="Requested
    URL=http://187.64.128.197/6ZKMSced/ Verb=POST RequestHeaders=POST /6ZKMSced/ HTTP/1.1 Referer:
    http://187.64.128.197/6ZKMSced/ Content-Type: multipart/form-data
    boundary=-----742816567139757 User-Agent: Mozilla/4.0 (compatible MSIE
    7.0 Windows NT 6.2 WOW64 Trident/7.0 .NET4.0C .NET4.0E .NET CLR 2.0.50727 .NET CLR 3.0.30729
    .NET CLR 3.5.30729) Host: 187.64.128.197 Content-Length: 4756 Connection: Keep-Alive
    Cache-Control: no-cache ResponseHeaders=HTTP/1.1 200 OK Server: nginx Date: Mon, 03 Aug 2020
    13:24:47 GMT Content-Type: text/html charset=UTF-8 Content-Length: 144884 Connection:
    keep-alive Vary: Accept-Encoding Status=200 Cache=Missed "
    URL="http://187.64.128.197/6ZKMSced/" Verb="POST" RequestHeaders="POST /6ZKMSced/ HTTP/1.1
    Referer: http://187.64.128.197/6ZKMSced/ Content-Type: multipart/form-data
    boundary=-----742816567139757 User-Agent: Mozilla/4.0 (compatible MSIE
    7.0 Windows NT 6.2 WOW64 Trident/7.0 .NET4.0C .NET4.0E .NET CLR 2.0.50727 .NET CLR 3.0.30729
    .NET CLR 3.5.30729) Host: 187.64.128.197 Content-Length: 4756 Connection: Keep-Alive
    Cache-Control: no-cache " ResponseHeaders="HTTP/1.1 200 OK Server: nginx Date: Mon, 03 Aug
```

```
2020 13:24:47 GMT Content-Type: text/html charset=UTF-8 Content-Length: 144884 Connection:
keep-alive Vary: Accept-Encoding " Status="200" UsageLogRequestCache="Missed"/>
```

Durch Prozess FXSRESM (PID: 2008) wird der Schlüsselwerteintrag (ValueName = FXSRESM) in den Registry-Run-Key Pfad der Windows-Registrierungsdatenbank vorgenommen (45, 46):

```
45 <Event MSec= "39771,0260" PID="2008" PName= "FXSRESM" TID="1788" EventName="EventID(1)"
    ProviderName="Microsoft-Windows-Kernel-Registry" BaseObject="0xffffe70269946d70"
    KeyObject="0xffffe70265547470" Status="0" Disposition="2" BaseName=""
    RelativeName="SOFTWARE\Microsoft\Windows\CurrentVersion\Run"/>
46 <Event MSec= "39771,5010" PID="2008" PName= "FXSRESM" TID="1788" EventName="EventID(5)"
    ProviderName="Microsoft-Windows-Kernel-Registry" KeyObject="0xffffe70265547470" Status="0"
    Type="1" DataSize="102" KeyName="" ValueName="FXSRESM" CapturedData="" PreviousDataType="0"
    PreviousDataSize="0" PreviousData=""/>
```

Es wird ein weiterer HTTP POST-Request für das Objekt (mjUb/ und optionalen Header an den Zielhost 187.64.128.197:80 übermittelt (47, 49, 50). Die Datei 3910.tmp wird zwischenzeitlich im Verzeichnis \Device\HarddiskVolume2\Users\JohnDoe\AppData\Local\Temp\ durch den Prozess FXSRESM (PID: 2008) erstellt (48):

```
47 <Event MSec= "39782,8451" PID="2008" PName= "FXSRESM" TID="1788"
    ActivityID="00cc000c000c0000d807fc06e0df8202" EventName="Wininet_SendRequest/Start"
    ProviderName="Microsoft-Windows-WinINet" Request="0x0282dfe0"
    AddressName="http://187.64.128.197/mjUb"/>
48 <Event MSec= "39786,7325" PID="2008" PName= "FXSRESM" TID="1104"
    ActivityID="1f3d28c269a70000b45d901fa769d601" EventName="CreateNewFile"
    ProviderName="Microsoft-Windows-Kernel-File" Irp="0xffffbf0b85cb6d28"
    FileObject="0xffffbf0b85c3cb80" IssuingThreadId="1.104" CreateOptions="33.554.528"
    CreateAttributes="128" ShareAccess="0"
    FileName="\Device\HarddiskVolume2\Users\JohnDoe\AppData\Local\Temp\3910.tmp"/>
49 <Event MSec= "39793,0695" PID="2008" PName= "FXSRESM" TID="1848"
    ActivityID="00cc000c000c0000d807fc06e0df8202" EventName="WININET_REQUEST_HEADER"
    ProviderName="Microsoft-Windows-WinINet" FormattedMessage="HTTP Request Headers
    RequestHandle=0x00cc000c POST /mjUb/ HTTP/1.1 Referer: http://187.64.128.197/mjUb/
    Content-Type: multipart/form-data boundary=-----456377212497216
    User-Agent: Mozilla/4.0 (compatible MSIE 7.0 Windows NT 6.2 WOW64 Trident/7.0 .NET4.0C
    .NET4.OE .NET CLR 2.0.50727 .NET CLR 3.0.30729 .NET CLR 3.5.30729) Host: 187.64.128.197
    Content-Length: 4756 Connection: Keep-Alive Cache-Control: no-cache "
    RequestHandle="0x00cc000c" Headers="POST /mjUb/ HTTP/1.1 Referer: http://187.64.128.197/mjUb/
    Content-Type: multipart/form-data boundary=-----456377212497216
    User-Agent: Mozilla/4.0 (compatible MSIE 7.0 Windows NT 6.2 WOW64 Trident/7.0 .NET4.0C
    .NET4.OE .NET CLR 2.0.50727 .NET CLR 3.0.30729 .NET CLR 3.5.30729) Host: 187.64.128.197
    Content-Length: 4756 Connection: Keep-Alive Cache-Control: no-cache "/>
50 <Event MSec= "39793,0818" PID="2008" PName= "FXSRESM" TID="1848"
    ActivityID="00cc000c000c0000d807fc06e0df8202" EventName="WININET_REQUEST_HEADER_OPTIONAL"
    ProviderName="Microsoft-Windows-WinINet" FormattedMessage="HTTP Request Headers OptionalData
    RequestHandle=0x00cc000c -----456377212497216 Content-Disposition:
    form-data name= TJxpwJBW filename= cUqPSxyGALxhUAjv Content-Type: application/octet-stream
    [Anm.: nicht darstellbar] "/>
51 <Event MSec= "39793,0928" PID="2008" PName= "FXSRESM" TID="1848"
    ActivityID="00cc000c000c0000d807fc06e0df8202" EventName="EventID(2002)"
    ProviderName="Microsoft-Windows-WinINet-Capture" FormattedMessage="The WinINet request payload
```



```

58 <Event MSec= "39879,3678" PID="2008" PName= "FXSRESM" TID="1104" EventName="FileIO/Write"
    FileName="C:\Users\JohnDoe\AppData\Local\winml\FXSRESMoe.exe" Offset="262.144"
    IrpPtr="0xFFFFFBF0B85420818" FileObject="0xFFFFFBF0B8530D400" FileKey="0xFFFFFE70266EEF1B0"
    IoSize="17.920" IoFlags="0"/>
    
```

Der Prozess FXSRESMoe mit der PID 4328 wird mit dem vom Systemverzeichnis kopierten Image der Eingabeaufforderung (cmd.exe) aus dem Temp-Verzeichnis des Benutzers gestartet (59). Die Datei 3910.tmp wird vom Prozess FXSRESMoe (4328) geöffnet (60). Nach Abfragen der Windows-Registrierungsdatenbank, u.a. zu Registry-Pfaden von Microsoft-Outlook, terminiert der Prozess wieder (61).

```

59 <Event MSec= "39887,1021" PID="4328" PName="FXSRESMoe" TID= "-1" EventName="Process/Start"
    ProcessID="4.328" ParentID="2.008" ImageFileName="FXSRESMoe.exe"
    DirectoryTableBase="0x0DCFC000" Flags="None" SessionID="1" ExitStatus="0x00000103"
    UniqueProcessKey="0xFFFFFBF0B819EC080" CommandLine="
    C:\Users\JohnDoe\AppData\Local\winml\FXSRESMoe.exe
    C:\Users\JohnDoe\AppData\Local\Temp\3910.tmp "/>
60 <Event MSec= "39939,7411" PID="4328" PName="FXSRESMoe" TID= "868" EventName="FileIO/Create"
    IrpPtr="0xFFFFFBF0B85CB6D28" FileObject="0xFFFFFBF0B851A69E0"
    CreateOptions="FILE_ATTRIBUTE_ARCHIVE, FILE_ATTRIBUTE_DEVICE" CreateDisposition="OPEN_EXISTING"
    FileAttributes="0" ShareAccess="None" FileName="C:\Users\JohnDoe\AppData\Local\Temp\3910.tmp"/>
61 <Event MSec= "39943,0505" PID="4328" PName= "" TID= "-1" EventName="Process/Stop"
    ProcessID="4.328" ParentID="2.008" ImageFileName="FXSRESMoe.exe"
    DirectoryTableBase="0x0DCFC000" Flags="None" SessionID="1" ExitStatus="0x00000000"
    UniqueProcessKey="0xFFFFFBF0B819EC080" CommandLine="
    C:\Users\JohnDoe\AppData\Local\winml\FXSRESMoe.exe
    C:\Users\JohnDoe\AppData\Local\Temp\3910.tmp "/>
    
```

Die Datei 3910.tmp wird vom Prozess FXSRESM (PID: 2008) geöffnet und anschließend gelöscht (62, 63):

```

62 <Event MSec= "39949,5297" PID="2008" PName= "FXSRESM" TID="1104"
    ActivityID="1f3d28c269a70000a25f901fa769d601" EventName="Create"
    ProviderName="Microsoft-Windows-Kernel-File" Irp="0xffffbf0b85cb6d28"
    FileObject="0xffffbf0b84ead7b0" IssuingThreadId="1.104" CreateOptions="16.777.312"
    CreateAttributes="0" ShareAccess="1"
    FileName="\Device\HarddiskVolume2\Users\JohnDoe\AppData\Local\Temp\3910.tmp"/>
63 <Event MSec= "39949,8165" PID="2008" PName= "FXSRESM" TID="1104" EventName="FileIO/Delete"
    FileName="C:\Users\JohnDoe\AppData\Local\Temp\3910.tmp" IrpPtr="0xFFFFFBF0B85CB6D28"
    FileObject="0xFFFFFBF0B84EAD7B0" FileKey="0xFFFFFE7026239B170" ExtraInfo="0x00000001"
    InfoClass="64"/>
    
```

Durch den Prozess FXSRESM (2008) wird ein HTTP Post Request mit zusätzlichem optionalem Header an den Endpunkt 187.64.128.197 gesendet (64-65)

```

64 <Event MSec= "41113,0512" PID="2008" PName= "FXSRESM" TID="1848" ActivityID="00cc000c00120000d807
    fc06e0df8202" EventName="WININET_REQUEST_HEADER" ProviderName="Microsoft-Windows-WinINet"
    FormattedMessage="HTTP Request Headers RequestHandle=0x00cc000c POST /Lqwx/P6tFUT7dfzX7XP28/
    WVoLEWB/ HTTP/1.1 Referer: http://187.64.128.197/Lqwx/P6tFUT7dfzX7XP28/WVoLEWB/ Content-Type:
    multipart/form-data boundary=-----718900250282428 User-Agent:
    Mozilla/4.0 (compatible MSIE 7.0 Windows NT 6.2 WOW64 Trident/7.0 .NET4.0C .NET4.0E .NET CLR
    2.0.50727 .NET CLR 3.0.30729 .NET CLR 3.5.30729) Host: 187.64.128.197 Content-Length: 4756
    Connection: Keep-Alive Cache-Control: no-cache " RequestHandle="0x00cc000c" Headers="POST
    
```

```

/Lqwx/P6tFUT7dfzX7XP28/WVoLEWB/ HTTP/1.1 Referer:
http://187.64.128.197/Lqwx/P6tFUT7dfzX7XP28/WVoLEWB/ Content-Type: multipart/form-data
boundary=-----718900250282428 User-Agent: Mozilla/4.0 (compatible MSIE
7.0 Windows NT 6.2 WOW64 Trident/7.0 .NET4.0C .NET4.0E .NET CLR 2.0.50727 .NET CLR 3.0.30729
.NET CLR 3.5.30729) Host: 187.64.128.197 Content-Length: 4756 Connection: Keep-Alive
Cache-Control: no-cache "/>

```

```

65 <Event MSec= "41113,0647" PID="2008" PName= "FXSRESM" TID="1848"
ActivityID="00cc000c00120000d807fc06e0df8202" EventName="WININET_REQUEST_HEADER_OPTIONAL"
ProviderName="Microsoft-Windows-WinINet" FormattedMessage="HTTP Request Headers OptionalData
RequestHandle=0x00cc000c -----718900250282428 Content-Disposition:
form-data name= wSNlujiteuzdoU filename= TSamReNaubFgGuncPb Content-Type:
application/octet-stream [Anm.: nicht darstellbar] " RequestHandle="0x00cc000c" Headers="
-----718900250282428 Content-Disposition: form-data name=
wSNlujiteuzdoU filename= TSamReNaubFgGuncPb Content-Type: application/octet-stream [Anm.:
nicht darstellbar]"/>

```

Es wird die Datei 5B7D.tmp wird im Verzeichnis \Device\HarddiskVolume2\Users\JohnDoe\AppData\Local\Temp\ durch den Prozess FXSRESM (PID: 2008) angelegt (66) und kurz darauf wieder gelöscht (67):

```

66 <Event MSec= "48592,1795" PID="2008" PName= "FXSRESM" TID="3888"
ActivityID="1f3d28c269a700000055911fa769d601" EventName="CreateNewFile"
ProviderName="Microsoft-Windows-Kernel-File" Irp="0xffffbf0b85cb6d28"
FileObject="0xffffbf0b85193c50" IssuingThreadId="3.888" CreateOptions="33.554.528"
CreateAttributes="128" ShareAccess="0"
FileName="\Device\HarddiskVolume2\Users\JohnDoe\AppData\Local\Temp\5B7D.tmp"/>
67 <Event MSec= "48593,4543" PID="2008" PName= "FXSRESM" TID="3888" EventName="FileIO/Delete"
FileName="C:\Users\JohnDoe\AppData\Local\Temp\5B7D.tmp" IrpPtr="0xFFFFBF0B85CB6D28"
FileObject="0xFFFFBF0B85193C50" FileKey="0xFFFFE70266EEF1B0" ExtraInfo="0x00000001"
InfoClass="64"/>

```

Der Prozess FXSRESM mit der PID 2924 von Prozess FXSRESM (PID: 2008) erzeugt. Dieser Prozess wird mit dem Parameter /scomma und Dateipfad C:\Users\JohnDoe\AppData\Local\winml\FXSRESM.exe /scomma C:\Users\JohnDoe\AppData\Local\Temp\5B7D.tmp aufgerufen (68):

```

68 <Event MSec= "48606,0652" PID="2924" PName= "FXSRESM" TID= "-1" EventName="Process/Start"
ProcessID="2.924" ParentID="2.008" ImageFileName="FXSRESM.exe" DirectoryTableBase="0x86395000"
Flags="Wow64" SessionID="1" ExitStatus="0x00000103" UniqueProcessKey="0xFFFFBF0B857D1080"
CommandLine=" C:\Users\JohnDoe\AppData\Local\winml\FXSRESM.exe /scomma
C:\Users\JohnDoe\AppData\Local\Temp\5B7D.tmp "/>
69 <Event MSec= "48617,3304" PID="2924" PName= "FXSRESM" TID="5040" EventName="Image/Load"
ImageBase="0x00400000" ImageSize="0x0005C000" ImageChecksum="362.819"
TimeDateStamp="1.565.284.678" DefaultBase="0x0040000000046000"
FileName="C:\Users\JohnDoe\AppData\Local\winml\FXSRESM.exe"/>

```

Es werden Registry-Schlüssel zu Internet Einstellungen und -cache des Internet Explorers (70-73) abgerufen und Informationen (Profile und Verläufe) aus den installierten Webbrowsern Mozilla Firefox (74-77) und Google Chrome (78-80) aus dem AppData-Verzeichnis des Benutzers aufgerufen sowie von weiteren gängigen Webbrowsern und Anwendungen. Exemplarisch werden die zugehörigen Events hervorgehoben:

```

70 <Event MSec= "48693,4207" PID="2924" PName= "FXSRESM" TID="5040" EventName="FileIO/QueryInfo"
    FileName="C:\Users\JohnDoe\AppData\Local\Microsoft\Windows\WebCache\WebCacheV01.dat"
    IrpPtr="0xFFFFBF0B85CB6D28" FileObject="0xFFFFBF0B85C478F0" FileKey="0xFFFFE70261A0F170"
    ExtraInfo="0x00000000" InfoClass="4"/>
71 [...] "C:\Users\JohnDoe\AppData\Local\Microsoft\Windows\INetCookies" [...]
72 [...] FileName=" [...]AppData\Local\Microsoft\Windows\INetCache\Content.IE5" [...]
73 [...] FileName=" [...]AppData\Local\Microsoft\Windows\WebCache\WebCacheV01.dat" [...]
74 [...] FileName=" [...]AppData\Roaming\Mozilla\Firefox\profiles.ini" [...]
75 [...] FileName=" [...]AppData\Roaming\Mozilla\Firefox\
    Profiles\nfagah75.default-release\history.dat" [...]
76 [...] FileName=" [...]AppData\Roaming\Mozilla\Firefox\Profiles\
    nfagah75.default-release\places.sqlite" [...]
77 [...] FileName=" [...]AppData\Roaming\Mozilla\Firefox\Profiles\
    nfagah75.default-release\key4.db" [...]
78 [...] FileName=" [...]AppData\Local\Google\Chrome\User Data\Default\Web Data" [...]
79 [...] FileName=" [...]AppData\Local\Google\Chrome\User Data\Default\Web Data-journal" [...]
80 [...] FileName=" [...]AppData\Local\Google\Chrome\User Data\Default>Login Data" [...]
81 [...] FileName=" [...]AppData\Roaming\Opera Software\Opera Stable>Login Data" [...]
82 [...] FileName=" [...]AppData\Roaming\Apple Computer\Preferences\keychain.plist" [...]

```

Im Anschluss schreibt der Prozess FXSRESM (2924) Daten in die Datei 5B7D.tmp und terminiert (83-85):

```

83 <Event MSec= "48902,0344" PID="2924" PName= "FXSRESM" TID="5040" EventName="FileIO/Write"
    FileName="C:\Users\JohnDoe\AppData\Local\Temp\5B7D.tmp" Offset="0" IrpPtr="0xFFFFBF0B8526DCE8"
    FileObject="0xFFFFBF0B85C46950" FileKey="0xFFFFE7026012B840" IoSize="3" IoFlags="395.776"/>
84 <Event MSec= "48903,1856" PID="2924" PName= "FXSRESM" TID="5040" EventName="FileIO/Write"
    FileName="C:\Users\JohnDoe\AppData\Local\Temp\5B7D.tmp" Offset="119"
    IrpPtr="0xFFFFBF0B8526DCE8" FileObject="0xFFFFBF0B85C46950" FileKey="0xFFFFE7026012B840"
    IoSize="2" IoFlags="395.776"/>
85 <Event MSec= "48909,9573" PID="2924" PName= "" TID= "-1" EventName="Process/Stop"
    ProcessID="2.924" ParentID="2.008" ImageFileName="FXSRESM.exe" DirectoryTableBase="0x86395000"
    Flags="Wow64" SessionID="1" ExitStatus="0x00000000" UniqueProcessKey="0xFFFFBF0B857D1080"
    CommandLine=" C:\Users\JohnDoe\AppData\Local\winml\FXSRESM.exe /scomma
    C:\Users\JohnDoe\AppData\Local\Temp\5B7D.tmp "/>

```

Der Prozess 2008 bildet die Datei 5B7D.tmp im Speicher ab (86) und löscht diese anschließend auf dem Dateisystem (89). Hierbei kann beobachtet werden, dass ein HTTP POST-Request zu einem weiteren Server mit der IP-Adresse 104.236.52.89 auf Port 8080 durchgeführt wird.

```

86 <Event MSec= "48921,1590" PID="2008" PName= "FXSRESM" TID="3888" EventName="FileIO/MapFile"
    ViewBase="0x022B0000" FileKey="0xFFFFE7026012B840" MiscInfo="0x0001000000000000"
    ViewSize="0x00001000" ByteOffset="0x00000000"
    FileName="C:\Users\JohnDoe\AppData\Local\Temp\5B7D.tmp"/>
87 <Event MSec= "48924,1905" PID="2008" PName= "FXSRESM" TID="3888" EventName="FileIO/UnmapFile"
    ViewBase="0x022B0000" FileKey="0xFFFFE7026012B840" MiscInfo="0x0001000000000000"
    ViewSize="0x00001000" ByteOffset="0x00000000"
    FileName="C:\Users\JohnDoe\AppData\Local\Temp\5B7D.tmp"/>
88 <Event MSec= "48924,5019" PID="2008" PName= "FXSRESM" TID="3888" EventName="FileIO/Delete"
    FileName="C:\Users\JohnDoe\AppData\Local\Temp\5B7D.tmp" IrpPtr="0xFFFFBF0B8526DCE8"
    FileObject="0xFFFFBF0B85C45370" FileKey="0xFFFFE7026012B840" ExtraInfo="0x00000001"
    InfoClass="64"/>

```



```

89 <Event MSec= "48924,5387" PID="2008" PName= "FXSRESM" TID="3888" EventName="FileIo/DletePath"
    ProviderName="MSNT_SystemTrace" IrpPtr="0xffffbf0b8526dce8" FileObject="0xffffbf0b85c45370"
    FileKey="0xffffe7026012b840" ExtraInfo="0x00000000" TTID="3.888" InfoClass="64"
    FileName="\Device\HarddiskVolume2\Users\JohnDoe\AppData\Local\Temp\5B7D.tmp"/>
90 <Event MSec= "48933,9445" PID="2008" PName= "FXSRESM" TID="3888"
    ActivityID="00cc001800240000d807300f88318e02" EventName="Wininet_SendRequest/Start"
    ProviderName="Microsoft-Windows-WinINet" Request="0x028e3188"
    AddressName="http://104.236.52.89:8080/EijhhdGx/uufe9gRHE1/Fy36UuQi4j/" />
91 <Event MSec= "49078,8931" PID="2008" PName= "FXSRESM" TID="1848"
    ActivityID="00cc001800240000d807300f88318e02" EventName="EventID(2001)"
    ProviderName="Microsoft-Windows-WinINet-Capture" FormattedMessage="The WinINet request header
    buffer captured " SessionId="6" SequenceNumber="0" Flags="3"
    Payload="504F5354202F45696A68684447782F75..." />
92 <Event MSec= "49078,9115" PID="2008" PName= "FXSRESM" TID="1848"
    ActivityID="00cc001800240000d807300f88318e02" EventName="EventID(2002)"
    ProviderName="Microsoft-Windows-WinINet-Capture" FormattedMessage="The WinINet request payload
    buffer captured " SessionId="6" SequenceNumber="0" Flags="1"
    Payload="0D0A2D2D2D2D2D2D2D2D2D2D2D2D2D2D..." />
93 <Event MSec= "49438,5066" PID="2008" PName= "FXSRESM" TID="1848"
    ActivityID="00cc001800240000d807300f88318e02" EventName="EventID(2003)"
    ProviderName="Microsoft-Windows-WinINet-Capture" FormattedMessage="The WinINet response header
    buffer captured " SessionId="6" SequenceNumber="0" Flags="3"
    Payload="485454502F312E3120323030204F4B0D..." />

```

Der Endpunkt mit der IP-Adresse 187.64.128.197 wird über Port 80 für einen weiteren HTTP POST-Request kontaktiert, wie den Informationen aus dem Event Wininet_UsageLogRequest (94) entnommen werden kann:

```

94 <Event MSec= "50725,1343" PID="2008" PName= "FXSRESM" TID="1788"
    ActivityID="00cc000c001e0000d807fc0610d05403" EventName="Wininet_UsageLogRequest"
    ProviderName="Microsoft-Windows-WinINet" FormattedMessage="Requested
    URL=http://187.64.128.197/GEBHTn7EN77YaIPv/OoCXUg5/b3Pss0C7vnRLBg7o/yDp1ESi/8ks1SO/ Verb=POST
    RequestHeaders=POST /GEBHTn7EN77YaIPv/OoCXUg5/b3Pss0C7vnRLBg7o/yDp1ESi/ 8ks1SO/ HTTP/1.1
    Referer: http://187.64.128.197/GEBHTn7EN77YaIPv/OoCXUg5/b3Pss0C7vnRLBg7o/yDp1ESi/8ks1SO/
    Content-Type: multipart/form-data boundary=-----302277833981364
    User-Agent: Mozilla/4.0 (compatible MSIE 7.0 Windows NT 6.2 WOW64 Trident/7.0 .NET4.0C
    .NET4.OE .NET CLR 2.0.50727 .NET CLR 3.0.30729 .NET CLR 3.5.30729) Host: 187.64.128.197
    Content-Length: 4772 Connection: Keep-Alive Cache-Control: no-cache ResponseHeaders=HTTP/1.1
    200 OK Server: nginx Date: Mon, 03 Aug 2020 13:25:00 GMT Content-Type: text/html charset=UTF-8
    Content-Length: 98884 Connection: keep-alive Vary: Accept-Encoding Status=200 Cache=Missed "
    URL="http://187.64.128.197/GEBHTn7EN77YaIPv/OoCXUg5/b3Pss0C7vnRLBg7o/yDp1ESi/ 8ks1SO/"
    Verb="POST" RequestHeaders="POST /GEBHTn7EN77YaIPv/OoCXUg5/b3Pss0C7vnRLBg7o/yDp1ESi/8ks1SO/
    HTTP/1.1 Referer:
    http://187.64.128.197/GEBHTn7EN77YaIPv/OoCXUg5/b3Pss0C7vnRLBg7o/yDp1ESi/8ks1SO/ Content-Type:
    multipart/form-data boundary=-----302277833981364 User-Agent:
    Mozilla/4.0 (compatible MSIE 7.0 Windows NT 6.2 WOW64 Trident/7.0 .NET4.0C .NET4.OE .NET CLR
    2.0.50727 .NET CLR 3.0.30729 .NET CLR 3.5.30729) Host: 187.64.128.197 Content-Length: 4772
    Connection: Keep-Alive Cache-Control: no-cache " ResponseHeaders="HTTP/1.1 200 OK Server:
    nginx Date: Mon, 03 Aug 2020 13:25:00 GMT Content-Type: text/html charset=UTF-8
    Content-Length: 98884 Connection: keep-alive Vary: Accept-Encoding " Status="200"
    UsageLogRequestCache="Missed" />
95 <Event MSec= "49489,2174" PID="2008" PName= "FXSRESM" TID="1848"
    ActivityID="00cc000c001e0000d807fc0610d05403" EventName="Wininet_SendRequest/Stop"
    ProviderName="Microsoft-Windows-WinINet" Request="0x0354d010" StatusLine="HTTP/1.1 200 OK" />

```

Die Datei 63DB.tmp wird im Verzeichnis \Device\HarddiskVolume2\Users\JohnDoe\AppData\Local\Temp\ durch den Prozess FXSRESM (PID: 2008) angelegt (96) und kurz darauf wieder gelöscht (97):

```

96 <Event MSec= "50731,3908" PID="2008" PName= "FXSRESM" TID="6104"
    ActivityID="1f3d28c269a700019ecd871fa769d601" EventName="CreateNewFile"
    ProviderName="Microsoft-Windows-Kernel-File" Irp="0xffffbf0b8526dce8"
    FileObject="0xffffbf0b85c47a80" IssuingThreadId="6.104" CreateOptions="33.554.528"
    CreateAttributes="128" ShareAccess="0"
    FileName="\Device\HarddiskVolume2\Users\JohnDoe\AppData\Local\Temp\63DB.tmp"/>
97 <Event MSec= "50731,9869" PID="2008" PName= "FXSRESM" TID="6104" EventName="FileIO/Delete"
    FileName="C:\Users\JohnDoe\AppData\Local\Temp\63DB.tmp" IrpPtr="0xFFFFBF0B8526DCE8"
    FileObject="0xFFFFBF0B85C48250" FileKey="0xFFFFFE702644DA1B0" ExtraInfo="0x00000001"
    InfoClass="64"/>

```

Es wird ein weiterer Prozess durch PID 2008 erstellt (FXSRESM, PID: 6816) (98), der mit dem gleichen Image (FXSRESM.exe) aufgerufen wird und ebenfalls über den Parameter /scomma und dem Dateipfad zu C:\Users\JohnDoe\AppData\Local\Temp\63DB.tmp verfügt.

Der Prozess ruft aus der Registry (99-101, 103) und im Dateisystem (102) Informationen zu möglichen E-Mail Postfächern und Kalenderkontakten ab. Anschließend terminiert der Prozess (104):

```

98 <Event MSec= "50737,4528" PID="6816" PName= "FXSRESM" TID= "-1" EventName="Process/Start"
    ProcessID="6.816" ParentID="2.008" ImageFileName="FXSRESM.exe" DirectoryTableBase="0x0B18E000"
    Flags="Wow64" SessionID="1" ExitStatus="0x00000103" UniqueProcessKey="0xFFFFBF0B857D1080"
    CommandLine=" C:\Users\JohnDoe\AppData\Local\winml\FXSRESM.exe /scomma
    C:\Users\JohnDoe\AppData\Local\Temp\63DB.tmp "/>
99 <Event MSec= "50800,3425" PID="6816" PName= "FXSRESM" TID="1252" EventName="EventID(2)"
    ProviderName="Microsoft-Windows-Kernel-Registry" BaseObject="0xfffffe70268490d70"
    KeyObject="0x00000000" Status="-1.073.741.772" Disposition="0" BaseName=""
    RelativeName="Software\Google\Google Desktop\Mailboxes"/>
100 <Event MSec= "50803,7396" PID="6816" PName= "FXSRESM" TID="1252" EventName="Registry/Open"
    Status="0x00000000" KeyHandle="0xFFFFFE70260E36C30" ElapsedTimeMSec="0,3432000000298"
    KeyName="Software\IncrediMail\Identities" Index="0"/>
101 <Event MSec= "50802,4082" PID="6816" PName= "FXSRESM" TID="1252" EventName="EventID(2)"
    ProviderName="Microsoft-Windows-Kernel-Registry" BaseObject="0xfffffe70268490d70"
    KeyObject="0x00000000" Status="-1.073.741.772" Disposition="0" BaseName=""
    RelativeName="Software\Microsoft\Office\Outlook\OMI Account Manager\Accounts"/>
102 <Event MSec= "50964,8314" PID="6816" PName= "FXSRESM" TID="1252" EventName="FileIO/Cleanup"
    IrpPtr="0xFFFFBF0B85CB6D28" FileObject="0xFFFFBF0B851A4460" FileKey="0xFFFFFE70264643960"
    FileName="C:\Windows\SysWOW64\MailContactsCalendarSync"/>
103 <Event MSec= "50802,6563" PID="6816" PName= "FXSRESM" TID="1252" EventName="EventID(2)"
    ProviderName="Microsoft-Windows-Kernel-Registry" BaseObject="0xfffffe70268490d70"
    KeyObject="0x00000000" Status="-1.073.741.772" Disposition="0" BaseName=""
    RelativeName="Software\Microsoft\Office\15.0\Outlook\Profiles"/>
104 <Event MSec= "51237,1006" PID="6816" PName= "" TID= "-1" EventName="Process/Stop"
    ProcessID="6.816" ParentID="2.008" ImageFileName="FXSRESM.exe" DirectoryTableBase="0x0B18E000"
    Flags="Wow64" SessionID="1" ExitStatus="0x00000000" UniqueProcessKey="0xFFFFBF0B857D1080"
    CommandLine=" C:\Users\JohnDoe\AppData\Local\winml\FXSRESM.exe /scomma
    C:\Users\JohnDoe\AppData\Local\Temp\63DB.tmp "/>

```

Auf die Datei 63DB.tmp wird durch Prozess FXSRESM (2008) zugegriffen (105). Die Datei wird durch den Prozess vom Dateisystem gelöscht (106):

```

105 <Event MSec= "51245,0896" PID="2008" PName= "FXSRESM" TID="6104" EventName="FileIO/Create"
      IrpPtr="0xFFFFBFB0B821EED28" FileObject="0xFFFFBFB0B85193C50"
      CreateOptions="FILE_ATTRIBUTE_ARCHIVE, FILE_ATTRIBUTE_DEVICE" CreateDisposition="CREATE_NEW"
      FileAttributes="0" ShareAccess="Read" FileName="C:\Users\JohnDoe\AppData\Local\Temp\63DB.tmp"/>
106 <Event MSec= "51245,4346" PID="2008" PName= "FXSRESM" TID="6104" EventName="FileIO/Delete"
      FileName="C:\Users\JohnDoe\AppData\Local\Temp\63DB.tmp" IrpPtr="0xFFFFBFB0B821EED28"
      FileObject="0xFFFFBFB0B8530D400" FileKey="0xFFFFE702644DA1B0" ExtraInfo="0x00000001"
      InfoClass="64"/>

```

Der ursprünglich vom Prozess 751 (PID: 6188) erzeugte Prozess FXSRESM (PID: 2008) terminierte während der Analyse nicht und führt weitere HTTP-Requests (107) und Response Anfragen zum Server 187.64.128.197 durch:

```

107 <Event MSec= "51588,4611" PID="2008" PName= "FXSRESM" TID="1788"
      ActivityID="00cc000c002a0000d807fc06d8718202" EventName="Wininet_UsageLogRequest"
      ProviderName="Microsoft-Windows-WinINet" FormattedMessage="Requested
      URL=http://187.64.128.197/8515XCwkJvXSF/VDeKADesot4L328/lgNTSY/49tJDCPeIs/ Verb=POST
      RequestHeaders=POST /8515XCwkJvXSF/VDeKADesot4L328/lgNTSY/49tJDCPeIs/ HTTP/1.1 Referer:
      http://187.64.128.197/8515XCwkJvXSF/VDeKADesot4L328/lgNTSY/49tJDCPeIs/ Content-Type:
      multipart/form-data boundary=-----370336456805357 User-Agent:
      Mozilla/4.0 (compatible MSIE 7.0 Windows NT 6.2 WOW64 Trident/7.0 .NET4.0C .NET4.0E .NET CLR
      2.0.50727 .NET CLR 3.0.30729 .NET CLR 3.5.30729) Host: 187.64.128.197 Content-Length: 4772
      Connection: Keep-Alive Cache-Control: no-cache ResponseHeaders=HTTP/1.1 200 OK Server: nginx
      Date: Mon, 03 Aug 2020 13:25:02 GMT Content-Type: text/html charset=UTF-8 Content-Length: 132
      Connection: keep-alive Status=200 Cache=Missed "
      URL="http://187.64.128.197/8515XCwkJvXSF/VDeKADesot4L328/lgNTSY/49tJDCPeIs/" Verb="POST"
      RequestHeaders="POST /8515XCwkJvXSF/VDeKADesot4L328/lgNTSY/49tJDCPeIs/ HTTP/1.1 Referer:
      http://187.64.128.197/8515XCwkJvXSF/VDeKADesot4L328/lgNTSY/49tJDCPeIs/ Content-Type:
      multipart/form-data boundary=-----370336456805357 User-Agent:
      Mozilla/4.0 (compatible MSIE 7.0 Windows NT 6.2 WOW64 Trident/7.0 .NET4.0C .NET4.0E .NET CLR
      2.0.50727 .NET CLR 3.0.30729 .NET CLR 3.5.30729) Host: 187.64.128.197 Content-Length: 4772
      Connection: Keep-Alive Cache-Control: no-cache " ResponseHeaders="HTTP/1.1 200 OK Server:
      nginx Date: Mon, 03 Aug 2020 13:25:02 GMT Content-Type: text/html charset=UTF-8
      Content-Length: 132 Connection: keep-alive " Status="200" UsageLogRequestCache="Missed"/>

```

6.3.6 Bewertung der Ergebnisse

Mit den aufgezeichneten Events aus ETW über den vollständigen Prozessbaum der Emotet-Infektion (siehe Bild 6.6) lässt sich das Verhalten ausgehend von der Makro-Ausführung bis einschließlich Kommunikation mit C2-Server und möglicher Exfiltration der durch den Trojaner Emotet ausgespähten Daten bestimmen und zeitlich rekonstruieren. Insgesamt wurden über einen Zeitraum von 20 Minuten und 32 Sekunden 1.334.142 Events aus 66 ETW-Providern (inkl. des Windows Kernel Traces) aufgezeichnet. Nachfolgende ETW-Provider lieferten forensisch relevante Events, die das Prozessverhalten aller durch Emotet und dem zugehörigen Loader erzeugten Verhaltensweisen darstellen:

1. Execution / Defense Evasion (T1106, T1086, T1064, T1047)

- Microsoft-Windows-Kernel-Process
- Microsoft-Windows-Kernel-Prefetch
- Microsoft-Windows-PowerShell
- Microsoft-Windows-Shell-Core
- Microsoft-Windows-DotNETRuntime
- Microsoft-Windows-RPC
- Windows Kernel (Keyword: Process, ImageLoad)

2. Persistence (T1060)

- Microsoft-Windows-Kernel-Registry

3. Discovery (T1012)

- Microsoft-Windows-Kernel-File
- Microsoft-Windows-Kernel-Registry
- Windows Kernel (Keyword: FileIO, Registry)

4. Lateral Movement / Command & Control (C2) (T1105)

- Microsoft-Windows-DNS-Client
- Microsoft-Windows-Winsock-AFD
- Microsoft-Windows-Winsock-NameResolution
- Microsoft-Windows-WinINet
- Microsoft-Windows-WinINet-Capture
- Microsoft-Windows-TCPIP
- Windows Kernel (Keyword: NetworkTCPIP)

Abweichend zum Analysebericht der Plattform ANY.RUN [ANY20b] konnten zusätzlich weitere unerwünschte und maliziöse Verhaltensweisen beobachtet werden. Hierbei konnte in hohem Detailgrad festgehalten werden, welche Aktivitäten durch den Emotet-Loader (Makrovirus und PowerShell-Ausführung) des Betriebssystems und der vorhandenen Komponenten (Windows Scripting Host, PowerShell, WinInet-Bibliothek und Windows Sockets) verwendet wurden, um das Analysesystem mit dem Trojaner zu infizieren und Kontakt zu entfernten Systemen aufzunehmen. Dabei konnte festgestellt werden, dass der Infektionsweg mit dem Analysebericht von ANY.RUN übereinstimmt. Über die Ausführung der obfuskierten PowerShell Befehle konnten über den ETW-Provider Microsoft-Windows-PowerShell Erkenntnisse über die aufgerufenen Methoden, Funktionen, Befehle und Skripte gewonnen werden, jedoch nicht über genaue Inhalte. Darüber hinaus konnte weiteres maliziöses Verhalten, wie das Ausspähen von Login- und Verlaufsdaten von Anwendungen, wie Mail-Clients

und Webbrowsern beobachtet werden. Hierzu wurden weitere Prozesse erstellt, die gezielt Informationen im System abgerufen haben. Eine Kommunikation hat mit drei Endpunkten stattgefunden:

1. 177.52.160.72:80 (Emotet-Download über den `WebClient`)
2. 187.64.128.197:80 (Command & Control (C2) über die WinINet-API)
3. 104.236.52.89:8080 (Command & Control (C2) über die WinINet-API)

Über die Endpunkte ist eine mögliche Exfiltration der vom Trojaner gesammelten Daten denkbar. Techniken der Taktik Exfiltration wurden nicht in der MITRE ATT&CK Matrix im Analysebericht von ANY.RUN [ANY20b] zugeordnet. Über die ETW-Provider Microsoft-Windows-WinINet und Microsoft-Windows-WinINet-Capture konnten Events aufgezeichnet werden, die POST-Requests erfassten, welche in zeitlichen Zusammenhang mit dem Auslesen von Registrierungsschlüsseln von Mailanwendungen und Webbrowsern standen. Die im Payload der Requests enthaltenen Daten lag nicht im Klartext vor und konnten nicht dekodiert werden, sodass Inhalte aus dem Payload der POST-Requests nicht eindeutig bestimmt werden konnten.

Durch die Ergebnisse in PoC III wird deutlich, dass das Verhalten einer Malware aufgrund abweichender Systemkonfiguration (Betriebssystem, vorhandene installierte Software) anders ausgeprägt sein kann. Auch Anti-Forensik Mechanismen einer Malware, wie die Detektion einer VM- oder Sandbox-Ausführung oder ein Debugging können Einfluss auf das Verhalten von Malware nehmen.

7 Grenzen und Herausforderungen

Der Einsatz von Event Tracing for Windows (ETW) zur Unterstützung forensischer Analysen von Prozessverhalten in Windows 10 ist mit Grenzen bei der Aufzeichnung als auch in einer späteren Auswertung verbunden. Zusätzlich sind Herausforderungen zu berücksichtigen, damit eine gezielte Prozessverhaltensanalyse mit ETW durchführbar ist.

Wesentliche Grenzen stellen vor allem die zur Verfügung stehenden Systemressourcen, wie die Prozessorleistung, der verfügbare Arbeitsspeicher oder der Sekundärspeicher eines Analysesystems dar. ETW ist als hochperformantes Framework im Windows Kernel dazu ausgelegt, ein Hilfsmittel für Softwareentwickler und Programmierer für einen Einsatz als Debugging-Werkzeug sowie zur Performanz- und Leistungsanalyse darzustellen. Bei einem Einsatz von ETW zu forensischen Analysen von Prozessverhalten ist daher zu berücksichtigen:

- Eine Vielzahl der zur Verfügung stehenden Provider bietet vorzugsweise Events über Start- und Endzeitpunkte bestimmter Aktivitäten, wie beispielsweise Zeitstempeln von I/O Zugriffen oder zur CPU-Nutzung (Profiling). Insbesondere bei längerer Aufzeichnungsdauer können sehr große Datenmengen aufgezeichnet werden. Aus Sicht der IT-Forensik in Bezug auf die Prozessverhaltensanalyse sind diese Informationen von nachrangiger Bedeutung und können eine Auswertung erschweren. Primär stehen Dateisystem-, Registry- oder Netzwerkzugriffe im Fokus, die sich durch diverse ETW-Provider teilweise sehr feingranular darstellen und protokollieren lassen. Die zugehörigen Events erfordern hierzu einen höheren Informationsgehalt als reine Zeitstempel.
- Für eine Malware-Analyse sind bei einer Prozessverhaltensanalyse aufgerufene API-Methoden (Syscalls) von besonderem Interesse. Diese lassen sich durch ETW zum aktuellen Zeitpunkt lediglich eingeschränkt erfassen. Die Events aus dem Windows Kernel Trace (über die NT Kernel Logger Session) mit dem Keyword *SystemCall* können nicht einem bestimmten Prozess zugeordnet werden, da der Event-Header keine PID beinhaltet. Da sowohl `SystemCall-Entry` als auch `SystemCall-Exit` als Event erfasst werden, wird eine sehr hohe Anzahl an

Events (>100.000/s) geliefert, die sich aus API-Aufrufen sämtlicher ausgeführter Prozesse ergibt [Sou12, S. 425]. Die Syscall-Aufrufe werden dabei nicht mit ihrem Funktionsnamen, sondern lediglich mit einer Adresse angegeben, die über alternative Methoden aufgelöst werden muss.

- Aus den aufgezeichneten Events der Dateisystem- und Registryoperationen können keine inhaltlichen Erkenntnisse aus Dateien oder Registrierungsschlüsselwerten gewonnen werden. (Verzeichnis-)Pfade und Art der entsprechenden Operation können mit ETW jedoch umfangreich aufgezeichnet und nachvollzogen werden. Zusätzliche Metainformationen zu Dateien, wie Dateigrößen oder zugehörige kryptografische Prüfsummen (Hashwerte) lassen sich den aufgezeichneten Events nicht entnehmen.
- Der Informationsgehalt bei der Aufzeichnung von Netzwerkoperationen variiert und ist von den vom Prozess genutzten Schnittstellen abhängig. Werden durch einen Prozess (System-)Programmierschnittstellen, wie beispielsweise die URL Monikers (`urlmon.dll`) oder die WinINet-API (`WinINet.dll`) verwendet, in denen systemseitig ETW-Provider implementiert sind, können über TCP/IP-Verbindungsinformationen hinaus, auch umfangreichere Informationen wie HTTP-Header und aufgerufene URLs entnommen werden.
- Es lässt sich immer nur dort ein bestimmtes Prozessverhalten durch ETW-Events erfassen, wenn entsprechende Schnittstellen auch Events an einen Provider liefern. Bestimmte Systemschnittstellen werden aufgrund der Betriebssystemkonzeption durch jeden Prozess durchlaufen, sodass bestimmte Events (z.B. zur Prozesserzeugung) stets aufgezeichnet werden können.

Die aufzuzeichnenden Informationen sind einzugrenzen, sodass alle forensisch relevanten Ereignisse und Aktivitäten über den zu untersuchenden Prozess erfasst werden, ohne das Analysesystem durch die Aufzeichnung zu überlasten. Eine Eingrenzung und Reduktion der aufzuzeichnenden Informationen lassen sich durch die gezielte Auswahl geeigneter ETW-Provider als Informationsquelle und Filterung prozessbezogener Ereignisse erreichen. In einer gezielten Prozessverhaltensanalyse mit ETW bestehen, neben den bereits erläuterten Grenzen, zusätzliche Herausforderungen:

- Software und Anwendungen können im Sinne des Multithreadings und Multiprocessings so gestaltet sein, dass sie über mehrere Prozesse hinweg ausgeführt werden (siehe Kapitel 3.1). Wie vor allem am Beispiel von PoC III: Emotet (Kapitel 6.3.) sichtbar wird, erstreckt sich auch eine Malware in der Regel über mehrere Prozesse, sodass die Aufzeichnung der Events aus den Providern

dynamisch auf erstellte Kindprozesse auszuweiten ist, um ein ganzheitliches Verhalten einer Software zu erfassen.

- Im konkreten Fall von PoC III wurde der erzeugte PowerShell-Prozess nicht als Kindprozess des initial zu überwachenden Prozess erzeugt. Alternative Möglichkeiten der Prozesserzeugung in Windows 10 oder auch Techniken wie Process Injection sollten daher ebenfalls berücksichtigt werden. Vor einer tatsächlichen Ausführung von (unbekannter) Software und Programmen ist nicht vorhersehbar, aus welchen ETW-Providern tatsächlich Events bereitgestellt werden (siehe Kapitel 6.1.3) und sich diese einem bestimmten Verhalten zuordnen lassen. Zur Laufzeit kann sich die Anzahl der ETW-Provider dynamisch verändern, beispielsweise wenn zusätzliche Bibliotheken eingebunden und weitere Funktionen verwendet werden. Die Aufzeichnung ist in diesem Fall dynamisch auf weitere ETW-Provider auszuweiten.
- Je nach ETW-Provider unterscheidet sich die Anzahl möglicher Events, die geliefert werden können. Dies erschwert eine Aggregation der Daten des Event-Payloads. Einheitlich verfügen alle Events lediglich über die vordefinierte Struktur des Event-Headers. Eine zu restriktiv gewählte Beschränkung der Aufzeichnung auf bestimmte Events kann dazu führen, dass bestimmte Events nicht mehr aufgezeichnet werden. Die Auswertung der Aufzeichnung wird durch nicht aussagekräftige Eventbezeichner (EventName) einiger ETW-Provider erschwert. Im Idealfall setzt sich der EventName aus Task und Opcode des Events zusammen (siehe Kapitel 3.9.4). Anstelle des EventName wird im vorgesehenen Header-Feld lediglich die EventID angegeben. Diese lässt sich mit den vorgestellten Hilfsmitteln in Kapitel 4.3.2 nachschlagen (z.B. ETWExplorer oder WEPEXplorer).
- Insbesondere bei einer hohen Anzahl aufgezeichneter Events kann sich eine Auswertung trotz Filterung nach Eventnamen und Prozess-ID als schwierig erweisen. Im Fall der Malware-Forensik ist nicht immer bestimmbar, ob es sich um legitimes oder schadhaftes Verhalten handelt. Zur Unterstützung der Auswertung sollten daher gängige Angreifertechniken oder Kompromittierungsindikatoren (engl. *indicator of compromise* / IoC) herangezogen werden und einen Ausgangspunkt für (Prozessverhaltens-)Analysen darstellen.
- Im Vergleich zur statischen Codeanalyse oder dem Reverse Engineering, bei dem sich potenziell der gesamte Funktionsumfang eines Programmes anhand des Programmcodes betrachten und erfassen lässt, kann durch eine Aufzeich-

nung der Events zu einem ausgeführten Prozess eines Programms mit Hilfe ETW lediglich der tatsächlich aufgerufene Funktionsumfang aufgezeichnet und ausgewertet werden.

Die ETW-API ist grundsätzlich für alle ausgeführten Prozesse zugänglich. Dies trifft auch auf Malware-Prozesse zu. ETW-Sitzungen können detailliert aufgelistet werden und mit administrativen Rechten beendet werden. Eine Malware könnte eine Prozessverhaltensanalyse mit ETW potenziell erkennen und das Verhalten anpassen oder die analysierende ETW-Sitzung beenden. Standardseitig werden einige Sitzungen jedoch immer ausgeführt. Eine Malware könnte daher gezielt die vom Standard abweichenden Sitzungen beenden und hierdurch eine Analyse erschweren.

8 Bewertung

Durch die Aufzeichnung von Events aus Providern des ETW-Frameworks lassen sich vielseitige und detaillierte Informationen in Windows 10 gewinnen, die sich grundsätzlich dazu eignen Prozessverhalten in Windows 10 forensisch zu analysieren und zu bestimmen.

Forensische Artefakte lassen sich sowohl aus bereitgestellten Events der im System registrierten ETW-Provider, als auch aus den Kernel-Providern über die NT Kernel Logger Sitzung im Rahmen einer Live-Forensik aufzeichnen und gewinnen. Aus den erzielten Erkenntnissen der durchgeführten Malware-Analysen in Kapitel 6, nach der in Kapitel 5 entwickelten Vorgehensweise, können zur Analyse von Prozessverhalten, Events über Aktivitäten und Verhaltensweisen aus den folgenden Kategorien erfasst und Prozessen zugeordnet werden:

- Prozesserzeugung und -terminierung
- Eingebundene Module und Bibliotheken in Prozessen
- Threadverhalten von Prozessen
- Dateisystemoperationen und -manipulationen
- Änderungen und Abfragen der Windows-Registrierungsdatenbank
- Netzwerk- und Interprozesskommunikation
- Skripting (z.B. Windows PowerShell)

Eine Auswahl forensisch relevanter Provider und eine Filterung von prozessbezogenen Events unterstützt und erleichtert die Aufzeichnung und spätere Auswertung, um eine Übersicht zu Verhaltensweisen und Aktivitäten von ausgeführten Prozessen zu erhalten. Bei der Aufzeichnung von Prozessen komplexer Software und Anwendungen, kann eine Reduzierung der aufzuzeichnenden Events notwendig sein, um das Prozessverhalten gezielt aufzeichnen und analysieren zu können.

Als Ausgangsgrundlage können Erkenntnisse bestehender Forschungsarbeiten sowie die Ergebnisse aus den durchgeführten Prozessverhaltensanalysen in Kapitel 6 (Proof-of-Concept) herangezogen werden, um relevante ETW-Provider aus Sicht der IT-Forensik für die Analysen von Prozessverhalten miteinzubeziehen:

- Microsoft-Windows-CAPI2
- Microsoft-Windows-DNS-Client
- Microsoft-Windows-DotNETRuntime
- Microsoft-Windows-FileInfoMinifilter
- Microsoft-Windows-Kernel-File
- Microsoft-Windows-Kernel-Prefetch
- Microsoft-Windows-Kernel-Process
- Microsoft-Windows-Kernel-Registry
- Microsoft-Windows-MPS-CLNT
- Microsoft-Windows-PowerShell
- Microsoft-Windows-RPC
- Microsoft-Windows-Shell-Core
- Microsoft-Windows-TCPIP
- Microsoft-Windows-URLMon
- Microsoft-Windows-WebIO
- Microsoft-Windows-WinINet
- Microsoft-Windows-WinINet-Capture
- Microsoft-Windows-Winsock-AFD
- Microsoft-Windows-Winsock-NameResolution

Neben den im System registrierten ETW-Providern liefern die Kernel Provider über die NT Kernel Logger Sitzung wichtige Erkenntnisse zum Prozessverhalten aus Sicht des Kernels. Anhang C stellt weiterführende Informationen zu den Kernel Keywords und den Keyword-Collections der TraceEvent Library zur Verfügung.

- Forensisch relevante Keywords des Windows Kernel Traces (NT Kernel Logger):
 - `process` (Process)
 - `thread` (Thread)
 - `img` (ImageLoad)
 - `registry` (Registry)
 - `fileio` (FileIO) / `fileiocompletion` (FileIOInit)
 - `net` (NetworkTCPIP)

Zu den hier aufgelisteten Providern werden im Anhang D relevante Events mit den beziehbaren Informationen und Daten detailliert aufgelistet. Diese Auswahl kann einerseits helfen, die aufzuzeichnenden Daten in zukünftigen Analysen zu reduzieren und andererseits eine Auswertung durch Filterung nach bestimmten Ereignissen und Aktivitäten gezielt zu unterstützen. Einige der in Kapitel 3.7 definierten Prozessver-

haltensweisen werden gegenwärtig nicht durch ETW erfasst. Forensisch relevante Prozessverhaltensweisen ließen sich in der Auswertung der Anwendungsfälle in Kapitel 6 beobachten und bestimmten Angriffstechniken und -taktiken nach MITRE zuordnen. Die Teilmenge der in dieser Arbeit identifizierten ETW-Provider ist nicht abschließend und kann durch weitere Analysen vergrößert werden. Auch die Konfiguration des Betriebssystems oder die installierte Software und Treiber können hierauf einen Einfluss nehmen.

Aufgezeichnete Events aus ETW beziehen sich immer auf das tatsächlich ausgeübte Verhalten des analysierten Prozesses. Sie garantieren keinen vollständigen Aufschluss über den möglichen Gesamtfunktionsumfang eines Programmes, wie mit Hilfe statischer Code-Analysen oder Reverse Engineering alternativ ermittelt werden könnte. Der Informationsgehalt von Events ist davon abhängig, welche Ereignisse und Aktivitäten der Softwareentwickelnde im Programmcode zur Erfassung mit ETW vorgesehen hat. Für das Betriebssystem Windows 10 und die integrierten Komponenten existieren ETW-Provider an wichtigen (System-)Programmierschnittstellen und Übergängen. Prozessverhaltensanalysen mit ETW können mit Black-Box-Tests kombiniert werden, um einen möglichst großen Verhaltensumfang einer (ausgeführten) Software zu erfassen. Der Aufwand von Black-Box-Tests zur gezielten Generierung von Events kann gegenüber einer statischen Code-Analyse oder dem Reverse-Engineering jedoch höher ausfallen.

Als Methode zur Prozessverhaltensanalyse bietet der Einsatz von ETW eine wertvolle Ergänzung zu den etablierten Verfahren und Vorgehensweisen einer dynamischen Malware-Forensik, wie beispielsweise dem Debugging. Im Vergleich zu statischen Analysemethoden können mit Hilfe von Prozessanalysen zusätzlich Verhaltensweisen erfasst werden, die sich erst aus weiterem nachgeladenem Programmcode dynamisch zur Laufzeit ergeben. Unterstützende forensische Analysen mit Hilfe von ETW liefern wichtige Erkenntnisse über das Prozessverhalten einer ausgeführten Malware und kommen dabei ohne die Integration zusätzlicher Komponenten aus, welche in der Regel einen tiefgehenden Systemeingriff oder auffällige Konfiguration erfordern. Die bei der Durchführung von Prozessverhaltensanalysen mit ETW erhobenen Informationen und Daten werden unmittelbar aus dem Betriebssystem über die ETW-API gewonnen. Dies schränkt die Detektionsmöglichkeiten im Rahmen von Anti-Forensik Mechanismen durch Malware erheblich ein. Hieraus ergibt sich ein Vorteil gegenüber konventionellen Werkzeugen, wie Debuggern, Sandboxes oder Netzwerkanalysertools, auf deren Erkennung gängige Malware üblicherweise ausgerichtet ist.

9 Zusammenfassung und Ausblick

In der vorliegenden Master-Thesis wurden Aufbau und Funktionsweise von ETW in Windows 10 mit ihren Schnittstellen betrachtet, sodass diese zur Analyse von Prozessverhalten unterstützend herangezogen werden können. Die beschriebenen Grenzen zeigen auf, dass einige Einschränkungen bei der Nutzung von ETW zu berücksichtigen sind und eine Prozessanalyse mit ETW nicht immer alle Verhaltensweisen vollständig abdeckt. Insbesondere für Malware, die Anti-Forensik Mechanismen einsetzt, um die Ausführung innerhalb einer Analyse-VM oder Sandbox zu erkennen, bietet ETW als integrierter Betriebssystemkomponente den Vorteil, dass es einer Malware erschwert wird die Analyse zu detektieren und ihr Verhalten anzupassen.

Ergebnisse aus bestehenden Forschungsarbeiten und Projekten, wie bspw. den vorgestellten Analysetools und -werkzeugen in Kapitel 4, dienen dieser Arbeit als Ausgangsgrundlage und konnten dazu herangezogen werden, um Prozesse hinsichtlich (maliziöser) Verhaltensweisen zu analysieren. Hierzu wurden alle zur Verfügung stehenden Informationsquellen (ETW-Provider) einbezogen, um später die forensisch relevanten Provider und Events identifizieren zu können. Insbesondere bei der Aufzeichnung bestimmter Prozesse wie von Office-Produkten oder der Windows PowerShell wurden eine hohe Anzahl von Events aus ETW-Providern aufgezeichnet, die sich nicht unmittelbar zur Bestimmung von Prozessverhaltensweisen heranziehen ließen. Vielmehr geben sie Aufschluss über die Performance oder sind als Debuggingmöglichkeit für Entwickler vorgesehen (z.B von .NET-Runtime Anwendungen).

In Kapitel 6 (PoC: Prozessanalysen) wurde gezeigt, dass sich clientseitig initiierte (HTTP-)Anfragen, wie GET und POST, durch den ETW-Provider Microsoft-Windows-WinINet in Windows 10 erfassen lassen, wenn die WinINet-Schnittstelle des Betriebssystems von einem Prozess verwendet wird. Handelt es sich beim angefragten Webserver um einen Microsoft Internet Information Services (IIS) Serverdienst, würden serverseitig Events über ETW-Provider, z.B. in Microsoft Windows Server 2016 oder Windows Server 2019 (ebenfalls Windows NT 10.0 Familie), Informationen aus Sicht des Dienstanbieters liefern. Durch weiterführende Analysen in unterschiedlichen Anwendungsfällen und Einsatzszenarien lassen sich potenziell weitere Provider

identifizieren. Hier können zusätzliche Dienst-spezifische ETW-Provider vor allem weitere forensische Informationen beim Einsatz von Infrastrukturdiensten wie den Active Directory Domain Services (AD DS) liefern.

ETW in Windows 10 wird stetig weiterentwickelt, sodass sich weitere Prozessverhaltensweisen aus den bereitgestellten Events aufzeichnen lassen. Das in der Thesis entwickelte Analysewerkzeug ETWProcessTracer sowie die vorgestellten Analysetools- und Werkzeuge in Kapitel 4.3.3 können hierbei unterstützend herangezogen werden.

Im Gegensatz zu klassischen ETW-Providern, die im System registriert sind und zu denen ein Manifest die zur Dekodierung benötigten Informationen beinhaltet, sind die sogenannten WPP-Provider für einen Einsatz zur Prozessverhaltensanalyse verhältnismäßig gering untersucht. Nach Matt Graeber [Gra19] stellen diese Provider eine weitere Form der forensischen Informationsgewinnung dar. Um an die erforderlichen Manifeste zu gelangen, bedarf es jedoch aufwendiger Verfahren wie des Reverse Engineerings von Software. Entwickler stellen die zum Event-Parsing benötigten Informationen (v.a. die privaten Symbole) in der Regel nicht öffentlich zur Verfügung. Durch weiterführende Forschung können WPP-Provider und die Provider des TraceLoggings untersucht werden, um zusätzliche forensische Informations- und Datenquellen zu identifizieren, die sich zur Prozessverhaltensanalyse unterstützend heranziehen lassen.

Neben einem Einsatz als Analysewerkzeug zu Forschungszwecken, ist eine Integration in Endpoint Detection and Response (EDR)-Tools oder Host Intrusion Prevention Systeme (HIPS) möglich, um festgelegte Events im System und Anwendungen nach bestimmten Heuristiken zu erfassen und auszuwerten. Durch eine permanente Überwachung bestimmter Komponenten, werden auffällige Verhaltensweisen nach vorgegebenen Erkennungsmustern aufgezeichnet, um auf bestimmte Ereignisse adäquat reagieren zu können. Durch eine geringe Integrationstiefe – im User-Space des Betriebssystems – lassen sich potenzielle Sicherheitsrisiken und Angriffsvektoren, die durch Einsatz solcher Software ausgehen, reduzieren.

Die erzielten Erkenntnisse aus der Prozessverhaltensanalyse mit Event Tracing for Windows (ETW) – insbesondere in der Malware-Forensik – können wichtige Anhaltspunkte zur Prävention und Detektion von Angriffen und Bedrohungen liefern und zur Reaktion auf einen erfolgreichen Angriff herangezogen werden.

Literaturverzeichnis

- [Air20a] Airbus CERT, Hrsg. *airbus-cert/etl-parser*. 02.06.2020. URL: <https://github.com/airbus-cert/etl-parser> (abgerufen am 30.06.2020).
- [Air20b] Airbus CERT, Hrsg. *airbus-cert/Winshark*. 23.07.2020. URL: <https://github.com/airbus-cert/Winshark> (abgerufen am 28.07.2020).
- [Anu20] Anurag. *njRAT Malware Analysis*. 21.06.2020. URL: <https://malwr-analysis.com/2020/06/21/njrat-malware-analysis/> (abgerufen am 15.07.2020).
- [ANY20a] ANY.RUN LLC., Hrsg. *123123.exe - Interactive analysis*. 16.06.2020. URL: <https://app.any.run/tasks/3001f041-bfa2-4abe-885e-7e6ec443f5e5/> (abgerufen am 15.07.2020).
- [ANY20b] ANY.RUN LLC., Hrsg. *Invoice_503_292647.doc - Interactive analysis*. 02.08.2020. URL: <https://app.any.run/tasks/2c4845fa-6ca6-409a-89bc-bf24ce8a0ab0/> (abgerufen am 06.08.2020).
- [Bac16] Elias Bachaalany. *Windows Events Providers Explorer*. 25.01.2016. URL: <https://lallouslab.net/2016/01/25/windows-events-providers-explorer/> (abgerufen am 29.06.2020).
- [Bau17] Christian Baun. *Betriebssysteme kompakt*. ger. IT kompakt. Berlin: Springer Vieweg, 2017. 267 S. ISBN: 9783662531426. URL: <http://www.springer.com/>.
- [Boo19a] Ruben Boonen. *fireeye/SilkETW*. Hrsg. von FireEye Corp. 30.12.2019. URL: <https://github.com/fireeye/SilkETW> (abgerufen am 15.06.2020).
- [Boo19b] Ruben Boonen. *SilkETW: Because Free Telemetry is ... Free!* Hrsg. von FireEye Corp. 20.03.2019. URL: <https://www.fireeye.com/blog/threat-research/2019/03/silketw-because-free-telemetry-is-free.html> (abgerufen am 15.06.2020).
- [Bro16] Zac Brown. *ETW Primer (KrabsETW)*. Hrsg. von Microsoft Corp. 27.10.2016. URL: <https://github.com/microsoft/krabsetw/blob/master/docs/EtwPrimer.md> (abgerufen am 29.06.2020).

- [Bro17a] Zac Brown. *Hidden Treasure: Intrusion Detection with ETW (Part 1)*. Hrsg. von Microsoft Corp. 11.04.2017. URL: <https://docs.microsoft.com/de-de/archive/blogs/office365security/hidden-treasure-intrusion-detection-with-etw-part-1> (abgerufen am 30.06.2020).
- [Bro17b] Zac Brown. *Hidden Treasure: Intrusion Detection with ETW (Part 2)*. Hrsg. von Microsoft Corp. 09.05.2017. URL: <https://docs.microsoft.com/de-de/archive/blogs/office365security/hidden-treasure-intrusion-detection-with-etw-part-2> (abgerufen am 30.06.2020).
- [Bro18] Zac Brown. *zacbrown/PowerKrabsEtw*. 21.02.2018. URL: <https://github.com/zacbrown/PowerKrabsEtw> (abgerufen am 29.06.2020).
- [Bun11] Bundesamt für Sicherheit in der Informationstechnik, Hrsg. *Leitfaden IT-Forensik*. 2011. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Cyber-Sicherheit/Themen/Leitfaden_IT-Forensik.pdf (abgerufen am 06.06.2020).
- [Bun18] Bundesamt für Sicherheit in der Informationstechnik, Hrsg. *Work Package 4: Telemetry*. 2018. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Cyber-Sicherheit/SiSyPHus/Workpackage4_Telemetry.pdf (abgerufen am 06.06.2020).
- [Bun20] Bundesamt für Sicherheit in der Informationstechnik, Hrsg. *Analyse der Telemetriekomponente in Windows 10. Konfigurations- und Protokollierungsempfehlung*. 2020. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Cyber-Sicherheit/SiSyPHus/Analyse_Telemetriekomponente_1_2.pdf (abgerufen am 06.06.2020).
- [BW20] Alexander Bode und Niels Warnars. *Detecting Fileless Malicious Behaviour of .NET C2Agents using ETW*. University of Amsterdam. 09.02.2020. URL: <https://delaat.net/rp/2019-2020/p56/report.pdf>.
- [Cyb16] CyberPoint International, Hrsg. *Logging Keystrokes with Event Tracing for Windows (ETW)*. 22.10.2016. URL: <https://www.cyberpointllc.com/srt/posts/srt-logging-keystrokes-with-event-tracing-for-windows-etw.html> (abgerufen am 30.06.2020).
- [Daw15] Bruce Dawson. *UIforETW – Windows Performance Made Easier*. 14.05.2015. URL: <https://randomascii.wordpress.com/2015/04/14/uiforetw-windows-performance-made-easier/> (abgerufen am 29.06.2020).
- [Fir17] FireEye Corp., Hrsg. *fireeye/pywintrace*. 16.11.2017. URL: <https://github.com/fireeye/pywintrace> (abgerufen am 30.06.2020).

- [Fre19] David French. *Detecting Adversary Tradecraft with Image Load Event Logging and EQL*. Hrsg. von A Medium Corp. 16.08.2019. URL: <https://medium.com/threatpunter/detecting-adversary-tradecraft-with-image-load-event-logging-and-eql-8de93338c16> (abgerufen am 05.08.2020).
- [Goo18] Google LLC, Hrsg. *google/UIforETW*. 01.02.2018. URL: <https://github.com/google/UIforETW> (abgerufen am 29.06.2020).
- [Gra18] Matt Graeber. *Tampering with Windows Event Tracing: Background, Offense, and Defense*. Hrsg. von A Medium Corp. 24.12.2018. URL: <https://medium.com/palantir/tampering-with-windows-event-tracing-background-offense-and-defense-4be7ac62ac63> (abgerufen am 06.06.2020).
- [Gra19] Matt Graeber. *Data Source Analysis and Dynamic Windows RE using WPP and TraceLogging*. Hrsg. von A Medium Corp. 22.02.2019. URL: <https://posts.specterops.io/data-source-analysis-and-dynamic-windows-re-using-wpp-and-tracelogging-e465f8b653f7> (abgerufen am 06.06.2020).
- [GZF12] Sasha Goldshtein, Dima Zurbalev und Ido Flatow. *Pro .NET performance. Optimize your C# applications ; make your users happier and your code snappier by maximizing the performance of your applications*. Books for professionals by professionals. New York, NY: Apress, 2012. 345 S. ISBN: 1430244585.
- [Has17] Matthew Hastings. *matthastings/PSalander*. 23.09.2017. URL: <https://github.com/matthastings/PSalander> (abgerufen am 30.06.2020).
- [Ibr18] Nicole Ibrahim. *ETW Event Tracing for Windows and ETL Files*. Hrsg. von Hacking Exposed Computer Forensics Blog. 07.06.2018. URL: <https://www.hecfblog.com/2018/06/etw-event-tracing-for-windows-and-etl.html> (abgerufen am 06.06.2020).
- [Jan17] Paula Januszkiewicz. *Tips on Advanced Monitoring & Alerting in Windows Systems / BeyondTrust*. Hrsg. von BeyondTrust Corp. 07.09.2017. URL: <https://www.beyondtrust.com/blog/entry/tips-advanced-monitoring-alerting-windows-systems> (abgerufen am 06.06.2020).
- [jdu20] jdu2600. *Windows10EtwEvents*. 17.05.2020. URL: <https://github.com/jdu2600/Windows10EtwEvents> (abgerufen am 17.08.2020).

- [Jur10] Mateusz Jurczyk. *Windows CSRSS Write Up: Inter-process Communication (part 1/3)*. 13.07.2010. URL: <https://j00ru.vexillium.org/2010/07/windows-csrss-write-up-inter-process-communication-part-1/> (abgerufen am 06.06.2020).
- [Kid19] Chrissy Kidd. *Tracing vs Logging vs Monitoring: What's the Difference?* Hrsg. von BMC Software, Inc. 07.03.2019. URL: <https://www.bmc.com/blogs/monitoring-logging-tracing/> (abgerufen am 06.06.2020).
- [LH18] Laimingas und Heidemarie Schuster. *Was ist Threat Hunting?* Hrsg. von Vogel IT-Medien GmbH. 03.09.2018. URL: <https://www.it-business.de/was-ist-threat-hunting-a-808131/> (abgerufen am 06.06.2020).
- [Mic07] Microsoft Corp., Hrsg. *LPC (Local procedure calls) Part 1 architecture*. 26.07.2007. URL: <https://docs.microsoft.com/de-de/archive/blogs/ntdebugging/lpc-local-procedure-calls-part-1-architecture> (abgerufen am 06.06.2020).
- [Mic16a] Microsoft Corp., Hrsg. *ETW Framework Conceptual Tutorial - Message Analyzer*. 26.10.2016. URL: <https://docs.microsoft.com/en-us/message-analyzer/etw-framework-conceptual-tutorial> (abgerufen am 06.06.2020).
- [Mic16b] Microsoft Corp., Hrsg. *Specifying Advanced ETW Session Configuration Settings - Message Analyzer*. 26.10.2016. URL: <https://docs.microsoft.com/en-us/message-analyzer/specifying-advanced-etw-session-configuration-settings> (abgerufen am 29.06.2020).
- [Mic16c] Microsoft Corp., Hrsg. *System ETW Provider Event Keyword-Level Settings - Message Analyzer*. 26.10.2016. URL: <https://docs.microsoft.com/en-us/message-analyzer/system-etw-provider-event-keyword-level-settings> (abgerufen am 29.06.2020).
- [Mic17a] Microsoft Corp., Hrsg. *Instrumenting Your Code with ETW*. 07.05.2017. URL: <https://docs.microsoft.com/en-us/windows-hardware/test/weg/instrumenting-your-code-with-etw> (abgerufen am 29.06.2020).
- [Mic17b] Microsoft Corp., Hrsg. *Memory-Mapped Files*. 30.3.2017. URL: <https://docs.microsoft.com/en-us/dotnet/standard/io/memory-mapped-files> (abgerufen am 06.06.2020).

- [Mic17c] Microsoft Corp., Hrsg. *Trace Log - Windows drivers*. 20.04.2017. URL: <https://docs.microsoft.com/en-us/windows-hardware/drivers/devtest/trace-log> (abgerufen am 29.06.2020).
- [Mic17d] Microsoft Corp., Hrsg. *tracert*. 16.10.2017. URL: https://docs.microsoft.com/de-de/windows-server/administration/windows-commands/tracert_1 (abgerufen am 06.06.2020).
- [Mic17e] Microsoft Corp., Hrsg. *wevtutil*. 16.10.2017. URL: <https://docs.microsoft.com/de-de/windows-server/administration/windows-commands/wevtutil> (abgerufen am 06.06.2020).
- [Mic18a] Microsoft Corp., Hrsg. *About Event Tracing - Win32 apps | Microsoft Docs*. 31.05.2018. URL: <https://docs.microsoft.com/en-us/windows/win32/etw/about-event-tracing> (abgerufen am 06.06.2020).
- [Mic18b] Microsoft Corp., Hrsg. *About Processes and Threads - Win32 apps | Microsoft Docs*. 31.05.2018. URL: <https://docs.microsoft.com/en-us/windows/win32/procthread/about-processes-and-threads> (abgerufen am 06.06.2020).
- [Mic18c] Microsoft Corp., Hrsg. *Anonymous Pipe Operations - Win32 apps*. 31.05.2018. URL: <https://docs.microsoft.com/en-us/windows/win32/ipc/anonymous-pipe-operations> (abgerufen am 06.06.2020).
- [Mic18d] Microsoft Corp., Hrsg. *Closing a File Mapping Object - Win32 apps*. 31.05.2018. URL: <https://docs.microsoft.com/en-us/windows/win32/memory/closing-a-file-mapping-object> (abgerufen am 06.06.2020).
- [Mic18e] Microsoft Corp., Hrsg. *Configuring and Starting a Private Logger Session - Win32 apps*. 31.05.2018. URL: <https://docs.microsoft.com/en-us/windows/win32/etw/configuring-and-starting-a-private-logger-session> (abgerufen am 29.06.2020).
- [Mic18f] Microsoft Corp., Hrsg. *Configuring and Starting an AutoLogger Session - Win32 apps*. 31.05.2018. URL: <https://docs.microsoft.com/en-us/windows/win32/etw/configuring-and-starting-an-autologger-session> (abgerufen am 27.08.2020).
- [Mic18g] Microsoft Corp., Hrsg. *Controlling Event Tracing Sessions*. 31.05.2018. URL: <https://docs.microsoft.com/de-de/windows/win32/etw/controlling-event-tracing-sessions> (abgerufen am 02.07.2020).

- [Mic18h] Microsoft Corp., Hrsg. *Event Metadata Overview - Win32 apps*. 31.05.2018. URL: <https://docs.microsoft.com/en-us/windows/win32/etw/event-metadata-overview> (abgerufen am 29.06.2020).
- [Mic18i] Microsoft Corp., Hrsg. *Event Tracing - Win32 apps*. 31.05.2018. URL: <https://docs.microsoft.com/en-us/windows/win32/etw/event-tracing-portal> (abgerufen am 06.06.2020).
- [Mic18j] Microsoft Corp., Hrsg. *Event Tracing Sessions - Win32 apps*. 31.05.2018. URL: <https://docs.microsoft.com/en-us/windows/win32/etw/event-tracing-sessions> (abgerufen am 29.06.2020).
- [Mic18k] Microsoft Corp., Hrsg. *How to: Use Logman to Collect Event Trace Data - Dynamics NAV*. 02.01.2018. URL: <https://docs.microsoft.com/en-us/dynamics-nav/how-to--use-logman-to-collect-event-trace-data> (abgerufen am 29.08.2020).
- [Mic18l] Microsoft Corp., Hrsg. *PowerShell*. 11.07.2018. URL: <https://docs.microsoft.com/de-de/windows-server/administration/windows-commands/powershell> (abgerufen am 06.06.2020).
- [Mic18m] Microsoft Corp., Hrsg. *Remote Procedure Call - Win32 apps*. 31.05.2018. URL: <https://docs.microsoft.com/en-us/windows/win32/rpc/rpc-start-page> (abgerufen am 29.06.2020).
- [Mic18n] Microsoft Corp., Hrsg. *Service Control Manager - Win32 apps*. 31.05.2018. URL: <https://docs.microsoft.com/en-us/windows/win32/services/service-control-manager> (abgerufen am 06.06.2020).
- [Mic18o] Microsoft Corp., Hrsg. *Terminating a Process - Win32 apps*. 31.05.2018. URL: <https://docs.microsoft.com/en-us/windows/win32/procthread/terminating-a-process> (abgerufen am 29.06.2020).
- [Mic18p] Microsoft Corp., Hrsg. *Windows Management Instrumentation - Win32 apps*. 31.05.2018. URL: <https://docs.microsoft.com/en-us/windows/win32/wmisdk/wmi-start-page> (abgerufen am 06.06.2020).
- [Mic19a] Microsoft Corp., Hrsg. *Configure Windows diagnostic data in your organization (Windows 10) - Windows Privacy*. 29.04.2019. URL: <https://docs.microsoft.com/en-us/windows/privacy/configure-windows-diagnostic-data-in-your-organization> (abgerufen am 06.06.2020).

- [Mic19b] Microsoft Corp., Hrsg. *The Microsoft.Diagnostics.Tracing.TraceEvent Library*. 15.10.2019. URL: <https://github.com/Microsoft/perfview/blob/master/documentation/TraceEvent/TraceEventLibrary.md> (abgerufen am 06.06.2020).
- [Mic20a] Microsoft Corp., Hrsg. *Microsoft Message Analyzer Blog*. 20.01.2020. URL: <https://docs.microsoft.com/en-us/openspecs/blog/ms-winintbloglp/dd98b93c-0a75-4eb0-b92e-e760c502394f> (abgerufen am 29.06.2020).
- [Mic20b] Microsoft Corp., Hrsg. *microsoft/etl2pcapng*. 13.05.2020. URL: <https://github.com/microsoft/etl2pcapng/blob/master/README.md> (abgerufen am 29.06.2020).
- [Mic20c] Microsoft Corp., Hrsg. *microsoft/krabsetw*. 22.05.2020. URL: <https://github.com/microsoft/krabsetw> (abgerufen am 29.06.2020).
- [Mic20d] Microsoft Corp., Hrsg. *Process Creation Flags (WinBase.h) - Win32 apps*. 27.07.2020. URL: <https://docs.microsoft.com/en-us/windows/win32/procthread/process-creation-flags> (abgerufen am 28.08.2020).
- [Mic20e] Microsoft Corp., Hrsg. *The TraceEvent Library Programmers Guide*. 18.03.2020. URL: <https://github.com/microsoft/perfview/blob/master/documentation/TraceEvent/TraceEventProgrammersGuide.md> (abgerufen am 29.06.2020).
- [Mic20f] Microsoft Corp., Hrsg. *Tools for Monitoring Performance Counters and Events*. 01.04.2020. URL: <https://docs.microsoft.com/en-us/dynamics365/business-central/dev-itpro/administration/tools-monitor-performance-counters-and-events> (abgerufen am 29.06.2020).
- [Mic20g] Microsoft Corp., Hrsg. *What is .NET TraceProcessing - .NET TraceProcessing*. 23.02.2020. URL: <https://docs.microsoft.com/en-us/windows/apps/trace-processing/overview> (abgerufen am 06.06.2020).
- [MIT18] MITRE Corp., Hrsg. *Data Encrypted, Technique T1022*. 17.10.2018. URL: <https://attack.mitre.org/techniques/T1022/> (abgerufen am 29.06.2020).
- [MIT20a] MITRE Corp., Hrsg. *Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder Technique T1060*. 25.03.2020. URL: <https://attack.mitre.org/techniques/T1060/> (abgerufen am 29.06.2020).

- [MIT20b] MITRE Corp., Hrsg. *MITRE ATT&CK Matrix for Enterprise*. 17.03.2020. URL: <https://attack.mitre.org/matrices/enterprise/> (abgerufen am 29.06.2020).
- [MIT20c] MITRE Corp., Hrsg. *Process Injection, Technique T1055*. 20.06.2020. URL: <https://attack.mitre.org/techniques/T1055/> (abgerufen am 29.06.2020).
- [MIT20d] MITRE Corp., Hrsg. *Remote File Copy, Technique T1105*. 17.06.2020. URL: <https://attack.mitre.org/techniques/T1105/> (abgerufen am 29.06.2020).
- [MIT20e] MITRE Corp., Hrsg. *Virtualization/Sandbox Evasion, Technique T1497*. 17.06.2020. URL: <https://attack.mitre.org/techniques/T1497/> (abgerufen am 29.06.2020).
- [Nag19] Luca Nagy. *VB2019 paper: Exploring Emotet, an elaborate everyday enigma*. Hrsg. von Virus Bulletin Ltd. SOPHOS Ltd. 2019. URL: <https://www.virusbulletin.com/virusbulletin/2019/10/vb2019-paper-exploring-emotet-elaborate-everyday-enigma/> (abgerufen am 10.08.2020).
- [nxl20] nxlog.co. *Solving Windows Log Collection Challenges with Event Tracing*. 27.01.2020. URL: <https://nxlog.co/whitepapers/windows-event-tracing> (abgerufen am 06.06.2020).
- [PB15] Insung Park und Alex Bendetovers. *Core OS Events in Windows 7, Part 1*. Hrsg. von Microsoft Corp. 15.08.2015. URL: <https://docs.microsoft.com/en-us/archive/msdn-magazine/2009/september/core-os-events-in-windows-7-part-1> (abgerufen am 24.08.2020).
- [PB19] Insung Park und Ricky Buch. *Event Tracing: Improve Debugging And Performance Tuning With ETW*. Hrsg. von Microsoft Corp. 30.09.2019. URL: <https://docs.microsoft.com/en-us/archive/msdn-magazine/2007/april/event-tracing-improve-debugging-and-performance-tuning-with-etw> (abgerufen am 06.06.2020).
- [R S20] R Software Inc, Hrsg. *What is an API Wrapper? | API Wrapper Definition | API Glossary. The Last Call - RapidAPI Blog*. 22.06.2020. URL: <https://rapidapi.com/blog/api-glossary/api-wrapper/> (abgerufen am 26.08.2020).

- [Rod19a] Roberto Rodriguez. *Threat Hunting with ETW events and HELK — Part 1: Installing SilkETW*. 19.09.2019. URL: <https://medium.com/threat-hunters-forge/threat-hunting-with-etw-events-and-helk-part-1-installing-silketw-6eb74815e4a0> (abgerufen am 06.06.2020).
- [Rod19b] Roberto Rodriguez. *Threat Hunting with ETW events and HELK — Part 2: Shipping ETW events to HELK*. 10.07.2019. URL: <https://medium.com/threat-hunters-forge/threat-hunting-with-etw-events-and-helk-part-2-shipping-etw-events-to-helk-16837116d2f5> (abgerufen am 06.06.2020).
- [Rod20] Roberto Rodriguez. *The Hunting ELK (HELK)*. 15.06.2020. URL: <https://github.com/Cyb3rWard0g/HELK> (abgerufen am 29.06.2020).
- [Rot17] Andy Rothman. *Windows Registry Attacks: Knowledge Is the Best Defense*. Hrsg. von Red Canary Inc. 20.04.2017. URL: <https://redcanary.com/blog/windows-registry-attacks-threat-detection/> (abgerufen am 06.06.2020).
- [Rus19] Mark Russinovich. *Process Monitor - Windows Sysinternals*. Hrsg. von Microsoft Corp. 18.12.2019. URL: <https://docs.microsoft.com/en-us/sysinternals/downloads/procmon> (abgerufen am 29.06.2020).
- [Sec15] Security Ninja. *Windows Functions in Malware Analysis - Cheat Sheet - Part 1*. 2015. URL: <https://resources.infosecinstitute.com/windows-functions-in-malware-analysis-cheat-sheet-part-1/> (abgerufen am 06.06.2020).
- [SH12] Michael Sikorski und Andrew Honig. *Practical malware analysis. The hands-on guide to dissecting malicious software*. San Francisco, Calif.: No Starch Press, 2012. 766 S. ISBN: 1-59327-290-1. URL: <http://www.loc.gov/catdir/enhancements/fy1206/2012000214-b.html>.
- [Sin15] Sushovon Sinha. *Physical and Virtual Memory in Windows 10*. 15.11.2015. URL: https://answers.microsoft.com/en-us/windows/forum/windows_10-performance/physical-and-virtual-memory-in-windows-10/e36fb5bc-9ac8-49af-951c-e7d39b979938 (abgerufen am 06.06.2020).
- [Sou12] Tarik Soulami. *Inside Windows debugging. Practical debugging and tracing strategies*. Sebastopol, Calif.: O'Reilly Media, 2012. 560 S. ISBN: 978-0-7356-6278-0.

- [Sro17] Pawel Srokosz. *Analysis of Emotet v4 - CERT Polska*. Hrsg. von CERT Polska. 24.05.2017. URL: <https://www.cert.pl/en/news/single/analysis-of-emotet-v4/> (abgerufen am 10.08.2020).
- [SS16] Ayman Shaaban und Konstantin Sapronov. *Practical Windows forensics. Leverage the power of digital forensics for Windows systems*. Community experience distilled. Birmingham, UK: Packt Publishing, 2016. ISBN: 978-1-78355-409-6. URL: <http://proquest.tech.safaribooksonline.de/9781783554096>.
- [TB15] Andrew S. Tanenbaum und Herbert Bos. *Modern operating systems*. 4th ed. Boston, MA: Prentice Hall und Pearson, 2015. 101 S. ISBN: 9780133591620.
- [Ten20] F. Tenzer. *Betriebssysteme - Marktanteile weltweit bis Januar 2020*. Hrsg. von Statista GmbH. 27.04.2020. URL: <https://de.statista.com/statistik/daten/studie/157902/umfrage/marktanteil-der-genutzten-betriebssysteme-weltweit-seit-2009/> (abgerufen am 07.06.2020).
- [Wol09] Kevin Woley. *FAQ: Common Questions for ETW and Windows Event Log*. 05.05.2009. URL: <https://social.msdn.microsoft.com/Forums/en-US/a1aa1350-41a0-4490-9ae3-9b4520aeb9d4/faq-common-questions-for-etw-and-windows-event-log> (abgerufen am 29.08.2020).
- [Yos+17] Pavel Yosifovich et al. *Windows internals*. 7th edition. Professional. Yosifovich, Pavel (Verfasser) Ionescu, Alex (Verfasser) Solomon, David A. (Verfasser) Russinovich, Mark E. (Verfasser). Redmond, Washington: Microsoft Press, 2017. 784 S. ISBN: 978-0-7356-8418-8.
- [Yos19a] Pavel Yosifovich. *zodiacon/EtwExplorer*. 29.12.2019. URL: <https://github.com/zodiacon/EtwExplorer> (abgerufen am 29.06.2020).
- [Yos19b] Pavel Yosifovich. *zodiacon/ProcMonX*. 03.03.2019. URL: <https://github.com/zodiacon/ProcMonX> (abgerufen am 29.06.2020).

Abbildungsverzeichnis

3.1	Hauptphasen der Prozesserstellung [Yos+17, S. 130]	15
3.2	Virtueller Speicher [Sin15]	15
3.3	Event Tracing for Windows [Mic18a]	25
3.4	Steuerungs- und Datenfluss in ETW (nach [Mic17a])	30
4.1	Verwalten von ETW-Sitzungen mit dem Performance Monitor	38
4.2	SilkETW im Betrieb als Windows-Dienst [Rod19b]	45
4.3	ETWProcessTracer (vereinfachte Darstellung der Funktionsweise)	48
5.1	Forensisches Vorgehensmodell bei der Analyse von Prozessverhalten mit ETW	56
6.1	Events des Providers Microsoft-Windows-DNS-Client (ETWExplorer)	73
6.2	PoC I - Prozessbaum über den DemoProcess	82
6.3	PoC II - Prozessbaum über die njRAT-Infektion	95
6.4	PoC II - njRat Registry-Run KeyValue	99
6.5	PoC III - Inhalt der Datei: Invoice_503_292647.doc	105
6.6	PoC III - Prozessbaum über die Emotet-Infektion	107

Listingverzeichnis

4.1	Verwalten von ETW-Sitzungen mit logman	38
6.1	DemoProcess - TV1 Abfragen von Registrierungsschlüsseln und -werten	67
6.2	DemoProcess - TV1 Setzen von Registrierungsschlüsseln	68
6.3	DemoProcess - TV2 Nachladen einer Datei von einem Webserver . . .	68
6.4	DemoProcess - TV3 Lesen- und Schreiben einer (Text-)Datei	69
6.5	DemoProcess - TV4 Erstellung eines Kindprozesses	69
6.6	Ausführung des DemoProcess mit dem ProcessStartHelper	71
6.7	Auszug aus der Auflistung aller im System registrierter ETW-Provider	71
6.8	Abfrage - Microsoft-Windows-DNS-Client Provider (Auszug)	72
6.9	PoC I - Abfrage von einem Prozess zugeordneten Provider (Auszug) .	73
6.10	PoC I - Abfrage der zugeordneten Provider vor TV1	74
6.11	PoC I - Abfrage der zugeordneten Provider nach TV4	75
6.12	PoC I - Abfrage zugeordneter Provider zur Laufzeit des Beispielprozesses	76
6.13	PoC I - Abfrage der zugeordneten Provider nach TV2	76
6.14	Abfrage der Kernel Keywords des Windows Kernel Traces	76
6.15	Instanziierung des TraceEventSession Objekts für die Analysesitzung	77
6.16	Zuordnung der im System registrierten Provider zur Analysesitzung .	78
6.17	Zuordnung des Kernel Traces zur Analysesitzung (nur Process)	79
6.18	Zuordnung des Kernel Traces zur Analysesitzung (alle Keywords) . .	79
6.19	Filterung prozessbezogener Events mit Hilfe von Callbacks	80
6.20	Erfassen von neu erstellten (Kind-)Prozessen durch Kernel-Trace Call- backs	80
6.21	PoC I - Laden des DemoProcess mit dem ProcessStartHelper	81
6.22	PoC I - Start der Analyse mit dem ETWProcessTracer	81
6.23	PoC I - Start des DemoProcess mit dem ProcessStartHelper	82
6.24	PoC I - Erfassung des gestarteten DemoProcess im ETWProcessTracer	82
6.25	PoC I - Erfassung des terminierten DemoProcess im ETWProcessTracer	83
6.26	PoC I - Information über das Ende der Aufzeichnung	83
6.27	PoC II - Informationen zum njRAT Malware-Sample	92
6.28	PoC II - Ermittlung der PID der Eingabeaufforderung (cmd.exe) . . .	93
6.29	PoC II - Start der Analyse mit dem ETWProcessTracer	93
6.30	PoC II - Ausführung des njRAT Malware-Samples 123123.exe	94
6.31	PoC II - Konsolenausgabe des ETWProcessTracers zu njRAT	94
6.32	PoC III - Informationen zum Emotet Malware-Sample	103
6.33	PoC III - Ermittlung der PID der Eingabeaufforderung (cmd.exe) . .	104
6.34	PoC III - Start der Analyse mit dem ETWProcessTracer	104
6.35	PoC III - Konsolenausgabe des ETWProcessTracers zu Emotet	105

Tabellenverzeichnis

3.1	Event-Log Kategorien in Windows 10 (nach [SS16, S. 162-167f])	23
3.2	Event-Deskriptoren (nach [Sou12, S. 442])	27
3.3	Parameter zur Providerkonfiguration (nach [Gra18])	28
4.1	Anzahl von ETW-Providern in ETW-Sitzungen der Windows-Telemetrie in Windows 10 1607 LTSB (nach: [Bun20, S. 10])	34
4.2	Gegenüberstellung der ETW-Analysewerkzeuge	48
6.1	Kopfzeile (Header) der CSV-Protokolldatei des ETWProcessTracers	83
C.1	Zuordnung der Kernel Keywords (ETW-API) zur TraceEvent Library	166
C.2	Kernel Keyword-Collections (TraceEvent Library)	167
C.3	Kernel Keyword-Collection: Forensic (ETWProcessTracer)	167
D.1	Microsoft-Windows-Kernel-Process (forensisch rel. Events)	168
D.2	Microsoft-Windows-Kernel-Prefetch (forensisch rel. Events)	168
D.3	Windows Kernel: Process / process (forensisch rel. Events)	168
D.4	Windows Kernel: ImageLoad / img (forensisch rel. Events)	169
D.5	Microsoft-Windows-DotNETRuntime (forensisch rel. Events)	169
D.6	Microsoft-Windows-Kernel-File (forensisch rel. Events)	169
D.7	Microsoft-Windows-FileInfoMinifilter (forensisch rel. Events)	169
D.8	Microsoft-Windows-Shell-Core (forensisch rel. Events)	169
D.9	Windows Kernel: FileIO / fileiocompletion (forensisch rel. Events)	170
D.10	Microsoft-Windows-Kernel-Registry (forensisch rel. Events)	170
D.11	Windows Kernel: Registry (forensisch rel. Events)	170
D.12	Microsoft-Windows-CAPI2 (forensisch rel. Events)	171
D.13	Microsoft-Windows-DNS-Client (forensisch rel. Events)	171
D.14	Microsoft-Windows-MPS-CLNT (forensisch rel. Events)	171
D.15	Microsoft-Windows-RPC (forensisch rel. Events)	172
D.16	Microsoft-Windows-TCPIP (forensisch rel. Events)	172
D.17	Microsoft-Windows-URLMon (forensisch rel. Events)	172
D.18	Microsoft-Windows-WebIO (forensisch rel. Events)	172
D.19	Microsoft-Windows-WinINet (forensisch rel. Events)	173
D.20	Microsoft-Windows-WinINet-Capture (forensisch rel. Events)	173
D.21	Microsoft-Windows-Winsock-AFD (forensisch rel. Events)	174
D.22	Microsoft-Windows-Winsock-NameResolution (forensisch rel. Events)	174
D.23	Windows Kernel: TcpIp (forensisch rel. Events)	174
D.24	Microsoft-Windows-PowerShell (forensisch rel. Events)	174

Abkürzungsverzeichnis

AD DS	A ctive D irectory D omain S ervices.
ADK	A ssessment and D eployment K it.
AFD	A ncillary F unction D river for WinSock.
API	A pplication P rogramming I nterface.
C2	C ommand and C ontrol.
CAPI	C rypto A PI.
CERT	C omputer E mergency R esponse T eam.
CLI	C ommand L ine I nterface.
COM	C omponent O bject M odell.
CSV	C omma-separated v alues.
DB4S	D B B rowser for S QLite.
DDE	D ynamic D ata E xchange.
DLL	D ynamic L ink L ibrary.
EDR	E ndpoint D etection and R esponse.
ELK	E lasticsearch L ogstash K ibana.
ETL	E vent T race L og.
ETW	E vent T racing for W indows.
EXE	E xecutable P rogram.
GUID	G lobally U nique I dentifier.
HIPS	H ost I ntrusion P revention S ystem.
I/O	I nput / O utput.
IIS	I nternet I nformation S ervices.
IoC	I ndicator of C ompromise.
IOCTL	I /O C ontrol.
IRP	I /O R equest P acket.
JSON	J ava S cript O bject N otation.
LPC	L ocal P rocedure C all.
MMA	M icrosoft M essage A nalyzer.
MOF	M anaged O bject F ormat.

PCAP	P acket C apture.
PCB	P rocess C ontrol B lock.
PDB	P rogramm D atabase.
PE	P ortable E xecutable.
PID	P rocess- ID (Process identifier).
PPID	P arent P rocess ID .
PS	Windows P ower S hell.
Ptr	P ointer.
RAT	R emote A ccess T rojaner.
RPC	R emote P rocedure C all.
SCM	S ervice C ontrol M anager.
SDK	S oftware D evelopment K it.
TID	T hread- ID (Thread identifier).
TLS	T ransport L ayer S ecurity.
TMF	T race M essage F ormat File.
UNC	U niform N aming C onvention.
URL	U niform R esource L ocator.
VBScript	V isual B asic S cript.
VM	V irtual M achine.
WinINet	W indows I nternet (API).
WMI	W indows M anagement I nstrumentation.
WmiPrvSE	W MI P rovider Host S ervice.
WPA	W indows P erformance A nalyzer.
WPP	W indows software trace P reprocessor.
WPR	W indows P erformance R ecorder.
WPT	W indows P erformance T oolkit.
XML	E xtensible M arkup L anguage.

A Auflistung von Quellcode

A.1 PoC-Analysewerkzeug: ETWProcessTracer (v.0.1)

```

/*
Author: M.Reuter
Version: 0.1
Last modified: 2020/08/10
5
EXPERIMENTAL CODE - For Educational and Research Purposes
USE AT YOUR OWN RISK! Do not use in production environments!

Used Packages:
10 -Package Microsoft.Diagnostics.Tracing.TraceEvent, Copyright (c) .NET Foundation and Contributors \n MIT license see:
    https://github.com/Microsoft/perfview/blob/master/LICENSE.TXT
-Package CommandLineParser, Copyright (c) 2005 - 2015 Giacomo Stelluti Scala & Contributors \n MIT license see:
    https://github.com/commandlineparser/commandline/blob/master/License.md

*/

15
using CommandLine;
using Microsoft.Diagnostics.Tracing;
using CommandLine.Text;
using System;

20 using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Microsoft.Diagnostics.Tracing.Session;
25 using Microsoft.Diagnostics.Tracing.Parsers;
using Microsoft.Diagnostics.Tracing.Etlx;
using System.IO;
using System.Threading;
using Microsoft.Diagnostics.Tracing.Parsers.Kernel;
30 using System.Dynamic;
using Dia2Lib;
using System.Diagnostics;

35 namespace ETWProcessTracer {
class Program {
private static int eventCounter = 0;
private static int timersec = 0;
private static int initPID = 999999999;
40 private static string providerList = "";
private static string outFolder = "";
private static List<string> pname;
private static List<string> spawnDetection;
private static KernelTraceEventParser.Keywords kernelKeywords;

45
static int Main(string[] args) {
Console.WriteLine("---[...]-");
Console.ForegroundColor = ConsoleColor.DarkRed;
Console.WriteLine("   xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx");
50 Console.WriteLine("   [PoC] EtwProcessTracer ");
Console.WriteLine("   v.0.1 (M. Reuter) ");
Console.WriteLine("   xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx");
Console.ResetColor();
Console.ForegroundColor = ConsoleColor.DarkGray;
55 Console.WriteLine("\nDescription:\nThis tool is part of Master-Thesis: \'Possibilities of using Event Tracing for Windows (ETW) to
    \nassist the Forensic Analysis of Process Behavior in Windows 10\' \n(Hochschule Wismar University of Applied Sciences
    Technology Business and Design, August 2020)");

```

```

Console.WriteLine("\nUsed packages: ");
Console.WriteLine("-Package Microsoft.Diagnostics.Tracing.TraceEvent, Copyright (c) .NET Foundation and Contributors \n MIT
    license see: https://github.com/Microsoft/perfview/blob/master/LICENSE.TXT");
Console.WriteLine("-Package CommandLineParser, Copyright (c) 2005 - 2015 Giacomo Stelluti Scala & Contributors \n MIT license see:
    https://github.com/commandlineparser/commandline/blob/master/License.md ");
60 Console.ResetColor();
Console.WriteLine("---[...]---\n");

var cmdArgs = Parser.Default.ParseArguments<CommandLineOptions>(args);
cmdArgs.WithParsed(
65     options => {
        Main(options);
    });

if (!(TraceEventSession.IsElevated() ?? false)) {
70     Console.ForegroundColor = ConsoleColor.Yellow;
    Console.WriteLine("[WARNING] User \"\" + Environment.UserDomainName + "\\\" + Environment.UserName + \"\" is not elevated. Starting
        ETW-Session requires Admin!");
    Console.ResetColor();
}

return 0;
75 }

static int Main(CommandLineOptions cmdArgs) {
    // Display config summary for review & set vars from args
    Console.WriteLine("CONFIGURATION:");
80     Console.WriteLine("___[...]___\n");
    if (cmdArgs.ArgProcessID != 0) {
        Console.WriteLine($"Target ProcessID: \t{cmdArgs.ArgProcessID}");
        initPID = cmdArgs.ArgProcessID;
    }
85     pname = cmdArgs.ArgPNames.ToList();
    Console.WriteLine($"Additional Process(es): \t");
    Console.WriteLine(String.Join(" ", pname));

90     spawnDetection = cmdArgs.ArgSpawnDetection.ToList();
    Console.WriteLine($"SpawnDetection: \t");
    Console.WriteLine(String.Join(" ", spawnDetection));

95     if (!String.IsNullOrEmpty(cmdArgs.ArgProviders)) {
        Console.WriteLine($"ProviderConfig: \t{cmdArgs.ArgProviders}");
        providerList = cmdArgs.ArgProviders;
    }
    else
100     Console.WriteLine($"ProviderConfig: \tall registered ETW-Providers");

    // Add KernelKeyword Parsers
    kernelKeywords = (KernelTraceEventParser.Keywords.Process);
    foreach (string k in cmdArgs.ArgKernelKeywords.ToList()) {
105         switch (k) {
            case "DiskFileIO":
                kernelKeywords |= KernelTraceEventParser.Keywords.DiskFileIO;
                break;
            case "DiskIO":
110                 kernelKeywords |= KernelTraceEventParser.Keywords.DiskIO;
                break;
            case "ImageLoad":
                kernelKeywords |= KernelTraceEventParser.Keywords.ImageLoad;
                break;
115                 case "MemoryHardFaults":
                    kernelKeywords |= KernelTraceEventParser.Keywords.MemoryHardFaults;
                    break;
            case "NetworkTCP/IP":
                kernelKeywords |= KernelTraceEventParser.Keywords.NetworkTCP/IP;
                break;
120                 case "ProcessCounters":
                    kernelKeywords |= KernelTraceEventParser.Keywords.ProcessCounters;
                    break;
            case "Profile":
125                 kernelKeywords |= KernelTraceEventParser.Keywords.Profile;
                    break;
            case "Thread":
                kernelKeywords |= KernelTraceEventParser.Keywords.Thread;

```

```

break;
130 case "ContextSwitch":
kernelKeywords |= KernelTraceEventParser.Keywords.ContextSwitch;
break;
case "DiskIOInit":
135 kernelKeywords |= KernelTraceEventParser.Keywords.DiskIOInit;
break;
case "Dispatcher":
kernelKeywords |= KernelTraceEventParser.Keywords.Dispatcher;
break;
140 case "FileIO":
kernelKeywords |= KernelTraceEventParser.Keywords.FileIO;
break;
case "FileIOInit":
kernelKeywords |= KernelTraceEventParser.Keywords.FileIOInit;
break;
145 case "Memory":
kernelKeywords |= KernelTraceEventParser.Keywords.Memory;
break;
case "Registry":
150 kernelKeywords |= KernelTraceEventParser.Keywords.Registry;
break;
case "SystemCall":
kernelKeywords |= KernelTraceEventParser.Keywords.SystemCall;
break;
155 case "VirtualAlloc":
kernelKeywords |= KernelTraceEventParser.Keywords.VirtualAlloc;
break;
case "VAMap":
kernelKeywords |= KernelTraceEventParser.Keywords.VAMap;
break;
160 case "AdvancedLocalProcedureCalls":
kernelKeywords |= KernelTraceEventParser.Keywords.AdvancedLocalProcedureCalls;
break;
case "DeferedProcedureCalls":
165 kernelKeywords |= KernelTraceEventParser.Keywords.DeferedProcedureCalls;
break;
case "Driver":
kernelKeywords |= KernelTraceEventParser.Keywords.Driver;
break;
170 case "Interrupt":
kernelKeywords |= KernelTraceEventParser.Keywords.Interrupt;
break;
case "SplitIO":
kernelKeywords |= KernelTraceEventParser.Keywords.SplitIO;
break;
175 case "ThreadTime":
kernelKeywords |= KernelTraceEventParser.Keywords.ThreadTime;
break;
case "Forensic":
kernelKeywords |= (KernelTraceEventParser.Keywords.Thread | KernelTraceEventParser.Keywords.ImageLoad |
180 KernelTraceEventParser.Keywords.NetworkTCPIP | KernelTraceEventParser.Keywords.FileIOInit |
KernelTraceEventParser.Keywords.Registry);
break;
case "All":
kernelKeywords |= KernelTraceEventParser.Keywords.All;
break;
185 case "Default":
kernelKeywords |= KernelTraceEventParser.Keywords.Default;
break;
case "Verbose":
kernelKeywords |= KernelTraceEventParser.Keywords.Verbose;
break;
190 case "OS":
kernelKeywords |= KernelTraceEventParser.Keywords.OS;
break;
default:
break;
195 }
}
Console.WriteLine($"KernelKeywords: \t{kernelKeywords}");

if (!String.IsNullOrEmpty(cmdArgs.ArgOutFolder)) {
200 outFolder = cmdArgs.ArgOutFolder;
Directory.CreateDirectory(outFolder);
if (!outFolder.Substring(outFolder.Length - 1).Equals("\\")) {

```



```

    outFolder += "\\";
}
205 }
else
    outFolder = Directory.GetCurrentDirectory() + "\\";

Console.WriteLine($"Logfile directory: {outFolder}");
210

if (cmdArgs.ArgTimer != 0) {
    Console.WriteLine($"Timer: {cmdArgs.ArgTimer}s");
    timersec = cmdArgs.ArgTimer;
}
215 else
    Console.WriteLine("Timer: not set");

Console.WriteLine("___[...]\n");

220 // Stop info and Start information
Console.ForegroundColor = ConsoleColor.White;
Console.WriteLine("\n[INFO] Press CTRL+C at any time to stop tracing");
Console.WriteLine("Press <Enter> to start monitoring...");
Console.ResetColor();
225 while (Console.ReadKey().Key != ConsoleKey.Enter) { }
Console.ForegroundColor = ConsoleColor.DarkGreen;
Console.WriteLine("\nProcess Trace monitoring started");
Console.ResetColor();

230 // Start analysis
Thread ETWAnalyzer = new Thread(ProcessUserKernelLive);
ETWAnalyzer.Start();

return 0;
235 }

private static void ProcessUserKernelLive() {
    // PID Watchlist
    List<int> pidWatchlist = new List<int> { initPID };
240 // PID SpawnDetection Watchlist
    List<int> spawnDetectPIDs = new List<int> { };

    using (TraceEventSession session = new TraceEventSession("ETWProcessTracerSession")) {
        // Session options
245 session.StopOnDispose = true;

        // CTRL+C Callback
        Console.CancelKeyPress += new ConsoleCancelEventHandler((object sender, ConsoleCancelEventArgs cancelArgs) => {
            Console.ForegroundColor = ConsoleColor.DarkRed;
250 Console.WriteLine("\n\n[CTRL+C] ETWProcessTracerSession will be terminated. Please wait...");
            Console.ResetColor();
            session.Dispose();
            cancelArgs.Cancel = true;
        });

255 // CSV-Output Log-File
        var csvFile = new StringBuilder();
        var csvFileName = outFolder + DateTime.Now.ToString("yyyy-MM-dd-HH:mm") + @"_LOG_PID-" + initPID + ".csv";
        var header = string.Format("{0};{1};{2};{3};{4};{5}", "ID", "Time", "PID", "ProviderName", "EventName", "EventMessage");
260 csvFile.AppendLine(header);

        // Enable KernelTrace Providers
        session.EnableKernelProvider(kernelKeywords);

265 // Enable UserTrace Providers
        if (!providerList.Equals("")) {
            // Initialize available ETW Providers
            string[] lines = File.ReadAllLines(providerList);
            string[] providersETW = new string[lines.Length];

270 // Prepare exported providers read from file
            for (int i = 0; i < lines.Length; i++)
                providersETW[i] = (lines[i].Substring(0, lines[i].IndexOf(" ") + 1)).TrimEnd();

275 // Enable UserTrace Providers (from file)
            foreach (string provider in providersETW) {
                if (!provider.Contains("{") {
                    try {

```



```

350 Console.WriteLine("[ " + data.TimeStamp.ToString("HH:mm:ss") + "] [ProcessName] [PROCESS CREATED] " + "PID: " + data.ProcessID +
    " (PPID: " + data.ParentID + ") ProcessName: \" + data.ProcessName + "\" ImagePath: \" + data.ImageFileName + "\" Args:
    \" + data.CommandLine + "\"");
}
else if (indexSpawnDetection != -1) { // SpawnDetection
spawnDetectPIDs.Add(data.ProcessID);

355 eventCounter++;
csvFile.AppendLine(prepareEvent(data));
printEventCounter();

Console.SetCursorPosition(0, Console.CursorTop);
360 Console.WriteLine("[ " + data.TimeStamp.ToString("HH:mm:ss") + "] [SpawnDetection] [PROCESS CREATED (SpawnDetection only)] " +
    "PID: " + data.ProcessID + " (PPID: " + data.ParentID + ") ProcessName: \" + data.ProcessName + "\" ImagePath: \" +
    data.ImageFileName + "\" Args: \" + data.CommandLine + "\"");
}
};

// Kernel Trace Callback for Process STOP events
365 session.Source.Kernel.ProcessStop += delegate (ProcessTraceData data) {
// Check if ProcessID of event header matches the watchlist
if (pidWatchlist.IndexOf(data.ProcessID) != -1 || pidWatchlist.IndexOf(data.ParentID) != -1 ||
    spawnDetectPIDs.IndexOf(data.ProcessID) != -1) {
// Remove PID of terminated process from Watchlist
pidWatchlist.Remove(data.ProcessID);
370 if (spawnDetectPIDs.IndexOf(data.ProcessID) != -1) {
spawnDetectPIDs.Remove(data.ProcessID);
Console.WriteLine("[ " + data.TimeStamp.ToString("HH:mm:ss") + "] " + " [SpawnDetection] Child spawn detection stopped for
    process (PID: " + data.ProcessID + " terminated)");
}
eventCounter++;
375 csvFile.AppendLine(prepareEvent(data));
printEventCounter();

Console.SetCursorPosition(0, Console.CursorTop);
Console.WriteLine("[ " + data.TimeStamp.ToString("HH:mm:ss") + "] " + " [PROCESS TERMINATED] " + "PID: " + data.ProcessID + "
    (PPID: " + data.ParentID + ") ProcessName: \" + data.ProcessName + "\" ImagePath: \" + data.ImageFileName + "\" Args:
    \" + data.CommandLine + "\"");
380 }
};

// User Trace - Callback (All Events)
385 session.Source.Dynamic.All += delegate (TraceEvent data) {
if (pidWatchlist.IndexOf(data.ProcessID) != -1) {
// Increase eventCounter
eventCounter++;
// Prepare event data and mark event for CSV export
csvFile.AppendLine(prepareEvent(data));
390 // Update eventCounter on command line
printEventCounter();
}
};

395 // Additional Callback for all unhandled events
session.Source.UnhandledEvents += delegate (TraceEvent data) {
if (pidWatchlist.IndexOf(data.ProcessID) != -1) {
eventCounter++;
csvFile.AppendLine(prepareEvent(data));
400 printEventCounter();
}
};

// Timer
405 if (timersec != 0) {
var timer = new Timer(delegate (object state) {
Console.ForegroundColor = ConsoleColor.DarkYellow;
Console.WriteLine("\n\n[TIMERINFO] ETWProcessTracer stopped after {0} sec\n[INFO] ETWProcessTracerSession will be terminated.
    Please wait...", timersec);
Console.ResetColor();
410 session.Source.StopProcessing();
}, null, timersec * 1000, Timeout.Infinite);

session.Source.Process();
File.WriteAllText(csvFileName, csvFile.ToString());
415 timer.Dispose();
}

```

```

else {
    session.Source.Process();
    File.WriteAllText(csvFileName, csvFile.ToString());
420 }
}
}

public class CommandLineOptions {
425 [Option("PID", Required = true, HelpText = "\nInitial root ProcessID to analyze. (Spawned child processes will be \nautomatically
    added to the watchlist (full analyze).\nExample: --PID 1234")]
    public int ArgProcessID { get; set; }

    [Option("ProcessName", Separator = ',', Required = false, HelpText = "\n(Optional) Additional list of processes to analyze (by
        name) \nPID(s) will be added automatically to the watchlist (full analyze) \nwhen the process (name) is spawned by any
        parent process(es) \nExample(s): --ProcessName powershell\n --ProcessName powershell,wmiprsve,msiexec,...")]
430 public IEnumerable<string> ArgPNames { get; set; }

    [Option("SpawnDetection", Separator = ',', Required = false, HelpText = "\n(Optional) Additional list of processes (by name) for
        spawn detection only (!) \nChild processes PID will be added automatically to the watchlist (full analyze)\nwhen a new
        process is spawned by the parent process.\nExample(s): --SpawnDetection wmiprsve\n --SpawnDetection
        powershell,wmiprsve,msiexec,...\nProcessName has a higher precedence (if a certain process name is specified \nin both
        options)")]
    public IEnumerable<string> ArgSpawnDetection { get; set; }

    [Option("ProviderConfig", Required = false, HelpText = "(Optional, Default: All available registered ETW-Providers will be
        enabled.)\nPath to list of (individual selected) ETW-Provider(s) \n(e.g. export of \"logman query providers\")\nFormat:
        ProviderName\t{GUID}\nExample: --ProviderConfig C:\\etw\\providers.txt")]
435 public string ArgProviders { get; set; }

    [Option("KernelKeywords", Separator = ',', Required = false, HelpText = "(Optional, Keyword: \"Process\" is always enabled by
        default.)\nDiskFileIO,DiskIO,ImageLoad,MemoryHardFaults,
        NetworkTCPIP,ProcessCounters,\nProfile,Thread,ContextSwitch,DiskIOInit,Dispatcher,FileIO,FileIOInit,
        Memory,\nRegistry,SystemCall,VirtualAlloc,VAMap,AdvancedLocalProcedureCalls,\nDeferedProcedureCalls,
        Driver,Interrupt,SplitIO\n\nForensic (Process,Thread,ImageLoad,NetworkTCPIP,FileIOInit,Registry)\nAll
        (Verbose,ContextSwitch,Dispatcher,FileIO,FileIOInit,Memory,Registry,\n\t\t\t\t\t VirtualAlloc,VAMap,SystemCall,OS)\nDefault
        (DiskIO,DiskFileIO,DiskIOInit,ImageLoad,MemoryHardFaults,NetworkTCPIP,\n Process,ProcessCounters,Profile,Thread) \nVerbose
        (Default,ContextSwitch,Dispatcher,FileIO,FileIOInit,Memory,Registry,\n VirtualAlloc,VAMap) \nOS
        (AdvancedLocalProcedureCalls,DeferedProcedureCalls,Driver,Interrupt,SplitIO) \nThreadTime
        (Default,ContextSwitch,Dispatcher)\n\nExamples: --KernelKeywords NetworkTCPIP,Registry,FileIOInit\n --KernelKeywords
        Forensic")]
    public IEnumerable<string> ArgKernelKeywords { get; set; }

440 [Option("dir", Required = false, HelpText = "(Optional) Change path of logfile out directory \nExample: --dir C:\\etw\\logs\\")
    public string ArgOutFolder { get; set; }

    [Option("timer", Required = false, Default = 0, HelpText = "\n(Optional) Set Timer (secs.) \nExample: --timer 30")]
    public int ArgTimer { get; set; }
445 }

private static string prepareEvent(TraceEvent data) {
    string datadump = data.ToString();
    datadump = datadump.Replace(System.Environment.NewLine, " ");
450 datadump = datadump.Replace(";", " ");
    datadump = datadump.Replace("&quot;", " ");

    var n = string.Format("{0};{1};{2};{3};{4};{5}", eventCounter.ToString(), data.TimeStamp.ToString("dd/MM/yyyy HH:mm:ss"),
        data.ProcessID.ToString(), data.ProviderName.ToString(), data.EventName, datadump);

455 return n;
}

private static void printEventCounter() {
    Console.SetCursorPosition(0, Console.CursorTop);
460 Console.Write("Events: " + eventCounter);
}
}
}

```

Listing A.1: Quellcode ETWProcessTracer v0.1

A.2 Exemplarischer Anwendungsfall: DemoProcess (PoC I)

```

/*
Author: M.Reuter
Version: 0.1
Last modified: 2020/07/29
5
EXPERIMENTAL CODE - For Educational and Research Purposes
USE AT YOUR OWN RISK! Do not use in production environments!

Code based on snippets from Programming reference for the Win32 API:
10 https://docs.microsoft.com/en-us/windows/win32/
*/
#include <iostream>
#include <sstream>
#include <windows.h>
15 #include <winreg.h>
#include <urlmon.h>
#include <fstream>
#include <wininet.h>
#include <time.h>
20
using namespace std;

#define BUFFER 8192 // getRegKeyVal
#pragma comment(lib,"URLMon.lib")
25
// proto (wrappers for win32 api useage)
string checkRegKeyExistsW32(HKEY hkey, LPCWSTR lpSubKey, DWORD ulOptions, REGSAM samDesired);
wstring getRegKeyValW32(HKEY hkey, LPCWSTR lpSubKey, LPCWSTR lpValue, DWORD ulOptions);
string createRegKeyValW32(HKEY hkey, LPCWSTR lpSubKey, LPCWSTR lpValue, LPCWSTR data, DWORD ulOptions);
30 string downloadFileW32(LPCWSTR url, LPCWSTR destination);
void createFileW32(LPCWSTR path);
void writeFileW32(LPCWSTR path, char text[]);
void readFileW32(LPCWSTR path);
void createChildProcessW32(LPCWSTR exe);
35 string getTime();
string getTimeFull();

int main(int argc, char* argv[])
{
40 // variables
stringstream pid;
pid << GetCurrentProcessId();
DWORD sleep = 0;

45 if (argc > 1)
    sleep = strtol(argv[1], nullptr, 0);
    cout << "-----" << endl;
    cout << "-----DEMO PROCESS-----" << endl;
    cout << "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" << endl;
    cout << "          Process ID: " << GetCurrentProcessId() << endl;
50    cout << "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" << endl;
    cout << "-----" << getTimeFull() << "-----" << endl;
    cout << "-----" << endl;
    system("pause");

55
    cout << "___[...]" << endl;
    cout << "\n[START]\t TV1: Abfragen und Setzen von Registrierungsschlüsseln und -werten " << "(" << getTime() << ")" << endl;
    cout << "___[...]" << endl;
    //system(("logman.exe query providers -pid " + pid.str() + "> START_TV1_providers.txt").c_str());
60 Sleep(1000);
    cout << getTime() << " [RegOpenKeyEx()] SOFTWARE\\Oracle\\VirtualBox Guest Additions " << checkRegKeyExistsW32(HKEY_LOCAL_MACHINE,
        L"SOFTWARE\\Oracle\\VirtualBox Guest Additions", 0, KEY_READ) << endl;
    Sleep(1000);
    wcout << getTime().c_str() << L" [RegQueryValueExW] HARDWARE\\Description\\System\\SystemBiosVersion " << "Wert: " <<
        getRegKeyValW32(HKEY_LOCAL_MACHINE, L"HARDWARE\\Description\\System", L"SystemBiosVersion", 0) << endl;
    Sleep(1000);
65    cout << getTime() << " [RegOpenKeyEx()] HARDWARE\\ACPI\\DSDT\\BOX_ " << checkRegKeyExistsW32(HKEY_LOCAL_MACHINE,
        L"HARDWARE\\ACPI\\DSDT\\BOX_", 0, KEY_READ) << endl;
    Sleep(1000);
    cout << getTime() << " [RegSetValueExW()] SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run; ValueName: TV1DemoRunKey; ValueData:
        C:\\TV1DemoRunKey.exe" << " Status: " << createRegKeyValW32(HKEY_CURRENT_USER,
        L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", L"TV1DemoRunKey", L"C:\\TV1DemoRunKey.exe", 0) << endl;
    cout << "---[...]" << endl;

```

```

70 //system("logman.exe query providers -pid " + pid.str() + "> ENDE_TV1_providers.txt").c_str());
cout << "[ENDE]\t TV1: Abfragen und Setzen von Registrierungsschlüsseln und -werten " << "(" << getTime() << ")" << endl;
Sleep(5000);

cout << "___[...]" << endl;
cout << "\n[START]\t TV2: Nachladen einer Datei von einem Webserver " << "(" << getTime() << ")" << endl;
75 cout << "___[...]" << endl;
//system("logman.exe query providers -pid " + pid.str() + "> START_TV2_providers.txt").c_str());
cout << getTime() << " [URLDownloadToFile()] https://fiw.hs-wismar.de/storages/hs-wismar/favicons/favicon_hsw_fiw.png,
TV2favicon.png" << " Status: " << downloadFileW32(L"https://fiw.hs-wismar.de/storages/hs-wismar/favicons/
favicon_hsw_fiw.png", L"TV2favicon.png") << endl;
Sleep(1000);
cout << "---[...]" << endl;
80 cout << "[ENDE]\t TV2: Nachladen einer Datei von einem Webserver " << "(" << getTime() << ")" << endl;
//system("logman.exe query providers -pid " + pid.str() + "> ENDE_TV2_providers.txt").c_str());
Sleep(5000);

cout << "___[...]" << endl;
85 cout << "\n[START]\t TV3: Lesen- und Schreiben einer (Text-) Datei " << "(" << getTime() << ")" << endl;
cout << "___[...]" << endl;
// system("logman.exe query providers -pid " + pid.str() + "> START_TV3_providers.txt").c_str());
cout << getTime() << " [CreateFile(), WriteFile()] Text \"TV3 TestText1234\" in TV3createFileTestFile.txt" << " Status: ";
writeFileW32(L"TV3CreateFileTestFile.txt", _strdup("TV3 TestText1234"));
90 Sleep(5000);
cout << getTime() << " [ReadFileEx()] CreateFileTestFile.txt" << " Status: ";
readFileW32(L"TV3CreateFileTestFile.txt");
Sleep(1000);
cout << "---[...]" << endl;
95 cout << "[Ende]\t TV3: Lesen- und Schreiben einer (Text-) Datei " << "(" << getTime() << ")" << endl;
// system("logman.exe query providers -pid " + pid.str() + "> ENDE_TV3_providers.txt").c_str());
Sleep(5000);

cout << "___[...]" << endl;
100 cout << "\n[START]\t TV4: Erstellen eines Kindprozesses " << "(" << getTime() << ")" << endl;
cout << "___[...]" << endl;
// system("logman.exe query providers -pid " + pid.str() + "> START_TV4_providers.txt").c_str());
cout << getTime() << " [CreateProcess()] EXE: C:\\Windows\\System32\\winver.exe" << " Status: ";
createChildProcessW32(L"C:\\Windows\\System32\\winver.exe");
105 cout << "---[...]" << endl;
cout << "[ENDE]\t TV4: Erstellen eines Kindprozesses " << "(" << getTime() << ")" << endl;
// system("logman.exe query providers -pid " + pid.str() + "> ENDE_TV4_providers.txt").c_str());
Sleep(5000);

110 cout << "\nTV1 - TV4 abgeschlossen. Beispielprozess kann beendet werden." << endl;
system("pause");

return 0;
}
115 // TV 1: Abfragen und Setzen von Registrierungsschlüsseln und -werten
string checkRegKeyExistsW32(HKEY hKey, LPCWSTR lpSubKey, DWORD ulOptions, REGSAM samDesired) {
// https://docs.microsoft.com/en-us/windows/win32/api/winreg/
HKEY phkResult;
LONG error = RegOpenKeyEx(hKey, lpSubKey, ulOptions, samDesired, &phkResult);
120 if (error != ERROR_SUCCESS) {
RegCloseKey(phkResult);
return "RegKey not found!";
}
else {
125 RegCloseKey(phkResult);
return "RegKey exists!";
}

return "Error!";
130 }
wstring getRegKeyValW32(HKEY hkey, LPCWSTR lpSubKey, LPCWSTR lpValue, DWORD ulOptions) {
// https://docs.microsoft.com/en-us/windows/win32/api/winreg/
HKEY phkResult;
WCHAR value[1024];
135 DWORD BufferSize = sizeof(value);

LONG error = RegOpenKeyEx(hkey, lpSubKey, ulOptions, KEY_READ, &phkResult);
if (error != ERROR_SUCCESS)
{
140 wprintf(L"%s could not be opened. Error code: %x\n", lpSubKey, &phkResult);
}
}

```

```

    ULONG errorR = RegQueryValueExW(phkResult, lpValue, 0, NULL, LPBYTE(value), &BufferSize);
    if (ERROR_SUCCESS == errorR) {
145     RegCloseKey(phkResult);
        return value;
    }
    RegCloseKey(phkResult);
}
150
return L"Not Found";
}
string createRegKeyValW32(HKEY hkey, LPCWSTR lpSubKey, LPCWSTR lpValue, LPCWSTR data, DWORD ulOptions) {
// https://docs.microsoft.com/en-us/windows/win32/api/winreg/
155 HKEY phkResult;
    if (RegOpenKeyExW(hkey, lpSubKey, ulOptions, KEY_ALL_ACCESS, &phkResult) != ERROR_SUCCESS)
    {
        cout << "ERROR: unable to open registry";
    }
160 if (RegSetValueExW(phkResult, lpValue, 0, REG_SZ, (LPBYTE)data, wcslen(data) * sizeof(TCHAR)) != ERROR_SUCCESS) {
    RegCloseKey(phkResult);
    return "ERROR";
}
else
165     return "OK";

return "";
}
// TV2: Nachladen einer Datei von einem Webserver
170 string downloadFileW32(LPCWSTR url, LPCWSTR destination) {
// https://docs.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/platform-apis/ms775123(v=vs.85)
    if (URLDownloadToFile(NULL, url, destination, 0, NULL) == S_OK)
        return "OK";
    else
175     return "FAILED";
}
// TV3: Lesen- und Schreiben einer (Text-) Datei
void createFileW32(LPCWSTR path) {
// https://docs.microsoft.com/en-us/windows/win32/fileio/ opening-a-file-for-reading-or-writing
180 HANDLE hFile;
    BOOL bFile;

    char chBuffer[] = "DemoProcessText";
    DWORD dwNoByteToWrite = strlen(chBuffer);
185     DWORD dwNoByteWritten = 0;

    hFile = CreateFile(path, GENERIC_READ | GENERIC_WRITE, FILE_SHARE_READ, NULL, CREATE_NEW, FILE_ATTRIBUTE_NORMAL, NULL);
    if (hFile == INVALID_HANDLE_VALUE) {
        cout << "CreateFile() failed. Error: " << GetLastError();
190         if (GetLastError() == 80)
            cout << " (file already exists)" << endl;
    }
    else
195     cout << "CreateFile() success." << endl;

    CloseHandle(hFile);
}
void writeFileW32(LPCWSTR path, char text[]) {
200 // https://docs.microsoft.com/en-us/windows/win32/fileio/ opening-a-file-for-reading-or-writing
    HANDLE hFile;
    char* DataBuffer = text;
    DWORD dwBytesToWrite = (DWORD)strlen(DataBuffer);
    DWORD dwBytesWritten = 0;
205     BOOL bErrorFlag = FALSE;

    hFile = CreateFile(path, GENERIC_READ | GENERIC_WRITE, FILE_SHARE_READ, NULL, CREATE_NEW, FILE_ATTRIBUTE_NORMAL, NULL);
    if (hFile == INVALID_HANDLE_VALUE) {
        cout << "CreateFile() failed. Error: " << GetLastError();
210         if (GetLastError() == 80)
            cout << " (file already exists)" << endl;
        CloseHandle(hFile);
        hFile = CreateFile(path, FILE_APPEND_DATA, 0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
    }
215     else
        cout << "CreateFile() success." << endl;

    bErrorFlag = WriteFile(hFile, DataBuffer, dwBytesToWrite, &dwBytesWritten, NULL);
}

```

```

220     if (FALSE == bErrorFlag)
        cout << "WriteFile() failed. Error: " << GetLastError();
        else {
            if (dwBytesWritten != dwBytesToWrite)
                printf("Error: dwBytesWritten != dwBytesToWrite\n");
225     }
        CloseHandle(hFile);
    }
}

void readFileW32(LPCWSTR path) {
    // https://docs.microsoft.com/en-us/windows/win32/fileio/ opening-a-file-for-reading-or-writing
230
    HANDLE hFile;
    BOOL bErrorFlag = FALSE;
    DWORD dwBytesRead = 0;
    char ReadBuffer[4192] = { 0 };
235    OVERLAPPED ol = { 0 };
    DWORD dwNoByteRead = 0;

    hFile = CreateFile(path, GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED, NULL);
    if (hFile != INVALID_HANDLE_VALUE) {
        bErrorFlag = ReadFileEx(hFile, ReadBuffer, sizeof(ReadBuffer) - 1, &ol, NULL);
240        if (bErrorFlag == FALSE)
            cout << "ReadFile Failed. Error: " << GetLastError() << endl;
            else
                cout << "ReadFile sucess. Value: " << ReadBuffer << endl;
245    }
    CloseHandle(hFile);
}

// TV4: Erstellen eines Kindprozesses
void createChildProcessW32(LPCWSTR exe) {
    // https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessa
250    STARTUPINFO sInfo;
    PROCESS_INFORMATION pInfo;

    ZeroMemory(&sInfo, sizeof(sInfo));
255    sInfo.cb = sizeof(sInfo);
    ZeroMemory(&pInfo, sizeof(pInfo));

    // Start the child process.
    if (!CreateProcess(exe, NULL, NULL, NULL, FALSE, 0, NULL, NULL, &sInfo, &pInfo)) {
260        printf("ERROR (%d).\n", GetLastError());
        return;
    }
    else
        cout << "OK. PID: " << GetProcessId(pInfo.hProcess) << endl;
265

    WaitForSingleObject(pInfo.hProcess, INFINITE);
    CloseHandle(pInfo.hProcess);
    CloseHandle(pInfo.hThread);
}

// Timestamps
string getTime() {
    char buffer[26];
    time_t timestamp;
275    struct tm tm_time;

    time(&timestamp);
    localtime_s(&tm_time, &timestamp);
    strftime(buffer, sizeof(buffer), "%X", &tm_time);

280    return buffer;
}

string getTimeFull() {
    char buffer[26];
    time_t timestamp;
285    struct tm tm_time;

    time(&timestamp);
    localtime_s(&tm_time, &timestamp);
    strftime(buffer, sizeof(buffer), "%d.%m.%Y %X", &tm_time);
290

    return buffer;
}
}

```

Listing A.2: Quellcode DemoProcess

A.3 Hilfstool: ProcessStartHelper

```

5  /*
   Author: M.Reuter
   Version: 0.1
   Date: 2020/06/22
6
7  EXPERIMENTAL CODE - For Educational and Research Purposes
8  USE AT YOUR OWN RISK! Do not use in production environments!
9
10 /*
11 #include <iostream>
12 #include <windows.h>
13
14 using namespace std;
15
16 int main(int argc, char* argv[])
17 {
18     LPCSTR path = "";
19
20     if (argc > 1) {
21         path = argv[1];
22     }
23     else {
24         cout << "Usage:\n\tProcessStartHelper.exe PEFile.[exe|scr|...]\n";
25         return 0;
26     }
27
28     cout << "ProcessStartHelper\nPID: " << GetCurrentProcessId() << "\n\nPress any key for loading " << path << " as suspended
29         process." << endl;
30     getchar();
31
32     // https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessa
33     STARTUPINFO si;
34     PROCESS_INFORMATION pi;
35     LPDWORD ex = 0;
36
37     ZeroMemory(&si, sizeof(si));
38     si.cb = sizeof(si);
39     ZeroMemory(&pi, sizeof(pi));
40
41     if (!CreateProcess(path, NULL, NULL, NULL, FALSE, CREATE_SUSPENDED, NULL, NULL, &si, &pi)) {
42         printf("ERROR (%d).\n", GetLastError());
43         return 0;
44     }
45     else
46         printf("OK: ");
47
48     cout << path << " assigned ProcessID: " << GetProcessId(pi.hProcess) << " [SUSPENDED] " << " \nPress <ENTER> to resume process" <<
49         endl;
50     getchar();
51     ResumeThread(pi.hThread);
52
53     return 0;
54 }

```

Listing A.3: Quellcode ProcessStartHelper

B Weitere Bilder

```

C:\> Command Prompt

-----DEMO PROCESS-----
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
      Process ID: 8108
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
-----01.08.2020 15:21:44-----
-----
Press any key to continue . . .

[START] TV1: Abfragen und Setzen von Registrierungsschlüsseln und -werten (15:21:49)
-----
15:21:53 [RegOpenKeyEx()] SOFTWARE\Oracle\VirtualBox Guest Additions RegKey not found!
15:21:54 [RegQueryValueExW] HARDWARE\Description\System\SystemBiosVersion Wert: ALASKA - 1072009
15:21:55 [RegOpenKeyEx()] HARDWARE\ACPI\DSDTBOX_ RegKey not found!
15:21:56 [RegSetValueExW()] SOFTWARE\Microsoft\Windows\CurrentVersion\Run; ValueName: TV1DemoRunKey; ValueDa
ta: C:\TV1DemoRunKey.exe Status: OK
-----
[ENDE] TV1: Abfragen und Setzen von Registrierungsschlüsseln und -werten (15:21:58)
-----

[START] TV2: Nachladen einer Datei von einem Webserver (15:22:03)
-----
15:22:05 [URLDownloadToFile()] https://fiw.hs-wismar.de/storages/hs-wismar/favicons/favicon_hsw_fiw.png, TV2
favicon.png Status: OK
-----
[ENDE] TV2: Nachladen einer Datei von einem Webserver (15:22:06)
-----

[START] TV3: Lesen- und Schreiben einer (Text-) Datei (15:22:15)
-----
15:22:19 [CreateFile(), WriteFile()] Text "TV3 TestText1234" in TV3CreateFileTestFile.txt Status: CreateFile
() success.
15:22:24 [ReadFileEx()] CreateFileTestFile.txt Status: ReadFile success. Value: TV3 TestText1234
-----
[Ende] TV3: Lesen- und Schreiben einer (Text-) Datei (15:22:25)
-----

[START] TV4: Erstellen eines Kindprozesses (15:22:34)
-----
15:22:38 [CreateProcess()] EXE: C:\WindowsSystem32\winver.exe Status: OK, PID: 9700
-----
[ENDE] TV4: Erstellen eines Kindprozesses (15:22:39)
-----

TV1 - TV4 abgeschlossen. Beispielprozess kann beendet werden.
Press any key to continue . . .

```

Bild B.1: Exemplarischer Anwendungsfall (PoC I: DemoProcess)

```

C:\> Command Prompt

Usage:
  ProcessStartHelper.exe PEFile.[exe|scr|...]

```

Bild B.2: Hilfstool - ProcessStartHelper

```

Administrator: Command Prompt
-----
[PoC] EtwProcessTracer
v.0.1 (M. Reuter)
-----

Description:
This tool is part of Master-Thesis: 'Possibilities of using Event Tracing for Windows (ETW) to
assist the Forensic Analysis of Process Behavior in Windows 10'
(Hochschule Wismar University of Applied Sciences Technology Business and Design, August 2020)

Used packages:
-Package Microsoft.Diagnostics.Tracing.TraceEvent, Copyright (c) .NET Foundation and Contributors
MIT license see: https://github.com/Microsoft/perfview/blob/master/LICENSE.TXT
-Package CommandLineParser, Copyright (c) 2005 - 2015 Giacomo Stelluti Scala & Contributors
MIT license see: https://github.com/commandlineparser/commandline/blob/master/License.md
-----

ETWProcessTracer 0.1.0.0

ERROR(S):
Required option 'PID' is missing.

--PID                Required.
Initial root ProcessID to analyze. (Spawned child processes will be
automatically added to the watchlist (full analyze).
Example: --PID 1234

--ProcessName        (Optional) Additional list of processes to analyze (by name)
PID(s) will be added automatically to the watchlist (full analyze)
when the process (name) is spawned by any parent process(es)
Example(s): --ProcessName powershell
--ProcessName powershell,wmiprvse,msiexec,...

--SpawnDetection     (Optional) Additional list of processes (by name) for spawn detection only (!)
Child processes PID will be added automatically to the watchlist (full analyze)
when a new process is spawned by the parent process.
Example(s): --SpawnDetection wmiprvse
--SpawnDetection powershell,wmiprvse,msiexec,...
ProcessName has a higher precedence (if a certain process name is specified
in both options)

--ProviderConfig     (Optional, Default: All available registered ETW-Providers will be enabled.)
Path to list of (individual selected) ETW-Provider(s)
(e.g. export of "logman query providers")
Format: ProviderName {GUID}
Example: --ProviderConfig C:\etw\providers.txt

--KernelKeywords     (Optional, Keyword: "Process" is always enabled by default.)
DiskFileIO,DiskIO,ImageLoad,MemoryHardFaults,NetworkTCP/IP,ProcessCounters,
Profile,Thread,ContextSwitch,DiskIOInit,Dispatcher,FileIO,FileIOInit,Memory,
Registry,SystemCall,VirtualAlloc,VAMap,AdvancedLocalProcedureCalls,
DeferredProcedureCalls,Driver,Interrupt,SplitIO

Forensic (Process,Thread,ImageLoad,NetworkTCP/IP,FileIOInit,Registry)
All (Verbose,ContextSwitch,Dispatcher,FileIO,FileIOInit,Memory,Registry,
VirtualAlloc,VAMap,SystemCall,OS)
Default (DiskIO,DiskFileIO,DiskIOInit,ImageLoad,MemoryHardFaults,NetworkTCP/IP,
Process,ProcessCounters,Profile,Thread)
Verbose (Default,ContextSwitch,Dispatcher,FileIO,FileIOInit,Memory,Registry,
VirtualAlloc,VAMap)
OS (AdvancedLocalProcedureCalls,DeferredProcedureCalls,Driver,Interrupt,SplitIO)
ThreadTime (Default,ContextSwitch,Dispatcher)

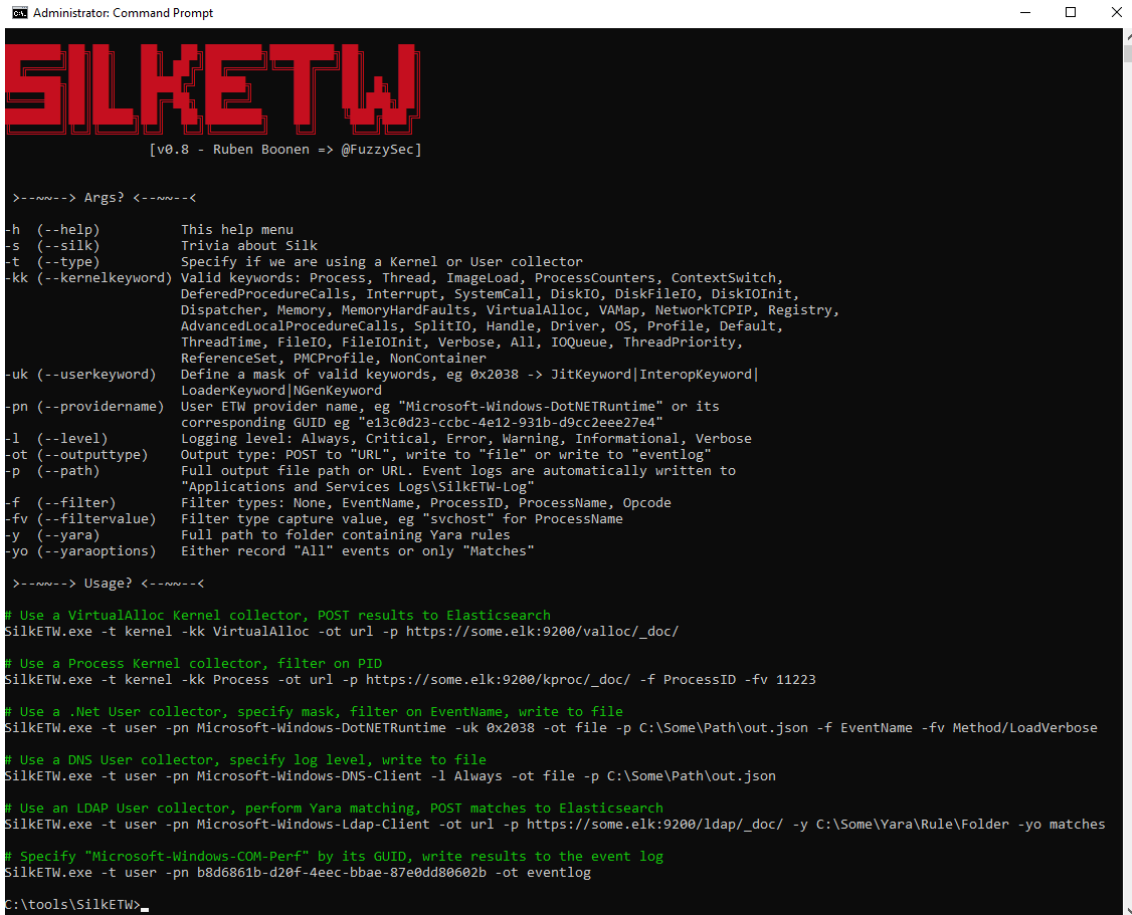
Examples: --KernelKeywords NetworkTCP/IP,Registry,FileIOInit
--KernelKeywords Forensic

--dir                (Optional) Change path of logfile out directory
Example: --dir C:\etw\logs\

--timer              (Default: 0)
(Optional) Set Timer (secs.)

```

Bild B.3: PoC-Analysewerkzeug - ETWProcessTracer (v.0.1)



```

Administrator: Command Prompt

SILKETW
[v0.8 - Ruben Boonen => @FuzzySec]

>-----> Angs? <-----<

-h (--help)          This help menu
-s (--silk)          Trivia about Silk
-t (--type)          Specify if we are using a Kernel or User collector
-kk (--kernelkeyword) Valid keywords: Process, Thread, ImageLoad, ProcessCounters, ContextSwitch,
                    DeferedProcedureCalls, Interrupt, SystemCall, DiskIO, DiskFileIO, DiskIOInit,
                    Dispatcher, Memory, MemoryHardFaults, VirtualAlloc, VAMap, NetworkTCPIP, Registry,
                    AdvancedLocalProcedureCalls, SplitIO, Handle, Driver, OS, Profile, Default,
                    ThreadTime, FileIO, FileIOInit, Verbose, All, IOQueue, ThreadPriority,
                    ReferenceSet, PMCPProfile, NonContainer
-uk (--userkeyword)  Define a mask of valid keywords, eg 0x2038 -> JitKeyword|InteropKeyword|
                    LoaderKeyword|NGenKeyword
-pn (--providername) User ETW provider name, eg "Microsoft-Windows-DotNETRuntime" or its
                    corresponding GUID eg "e13c0d23-ccbc-4e12-931b-d9cc2eee27e4"
-l (--level)          Logging level: Always, Critical, Error, Warning, Informational, Verbose
-ot (--outputtype)   Output type: POST to "URL", write to "file" or write to "eventlog"
-p (--path)          Full output file path or URL. Event logs are automatically written to
                    "Applications and Services Logs\SilkETW-Log"
-f (--filter)        Filter types: None, EventName, ProcessID, Opcode
-fv (--filtervalue)  Filter type capture value, eg "svchost" for ProcessName
-y (--yara)          Full path to folder containing Yara rules
-yo (--yaraoptions)  Either record "All" events or only "Matches"

>-----> Usage? <-----<

# Use a VirtualAlloc Kernel collector, POST results to Elasticsearch
SilkETW.exe -t kernel -kk VirtualAlloc -ot url -p https://some.elk:9200/valloc/_doc/

# Use a Process Kernel collector, filter on PID
SilkETW.exe -t kernel -kk Process -ot url -p https://some.elk:9200/kproc/_doc/ -f ProcessID -fv 11223

# Use a .Net User collector, specify mask, filter on EventName, write to file
SilkETW.exe -t user -pn Microsoft-Windows-DotNETRuntime -uk 0x2038 -ot file -p C:\SomePath\out.json -f EventName -fv Method/LoadVerbose

# Use a DNS User collector, specify log level, write to file
SilkETW.exe -t user -pn Microsoft-Windows-DNS-Client -l Always -ot file -p C:\SomePath\out.json

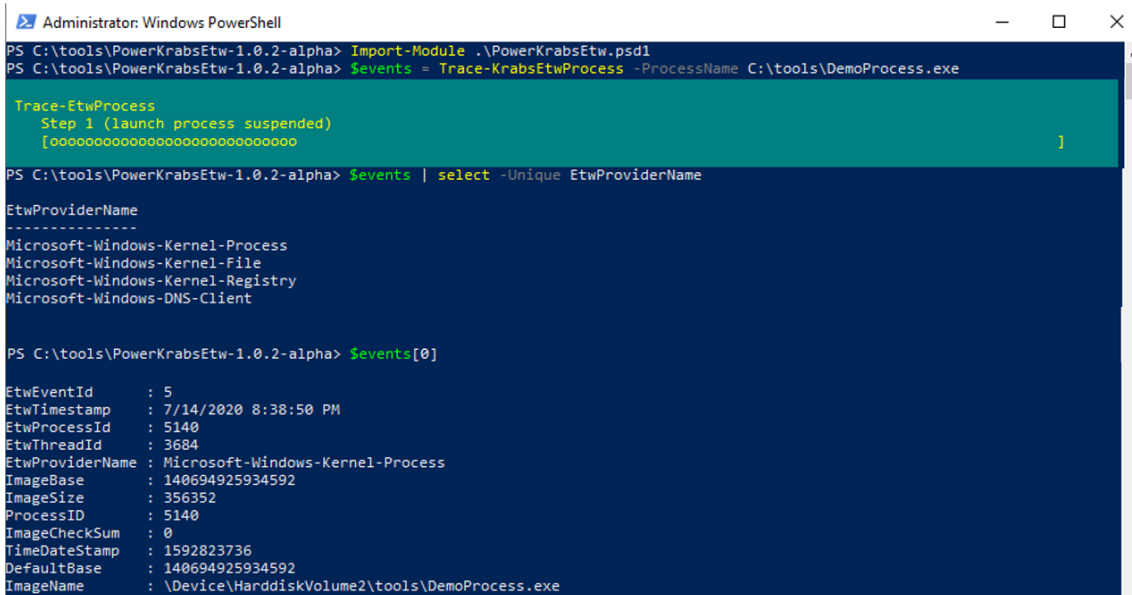
# Use an LDAP User collector, perform Yara matching, POST matches to Elasticsearch
SilkETW.exe -t user -pn Microsoft-Windows-Ldap-Client -ot url -p https://some.elk:9200/ldap/_doc/ -y C:\SomeYara\Rule\Folder -yo matches

# Specify "Microsoft-Windows-COM-Perf" by its GUID, write results to the event log
SilkETW.exe -t user -pn b8d6861b-d20f-4eec-bbae-87e0dd80602b -ot eventlog

C:\tools\SilkETW>

```

Bild B.4: SilkETW (v.0.8)



```

Administrator: Windows PowerShell

PS C:\tools\PowerKrabsEtw-1.0.2-alpha> Import-Module .\PowerKrabsEtw.psd1
PS C:\tools\PowerKrabsEtw-1.0.2-alpha> $events = Trace-KrabsEtwProcess -ProcessName C:\tools\DemoProcess.exe

Trace-EtwProcess
Step 1 (launch process suspended)
[ooooooooooooooooooooooooooooooooooooo]

PS C:\tools\PowerKrabsEtw-1.0.2-alpha> $events | select -Unique EtwProviderName

EtwProviderName
-----
Microsoft-Windows-Kernel-Process
Microsoft-Windows-Kernel-File
Microsoft-Windows-Kernel-Registry
Microsoft-Windows-DNS-Client

PS C:\tools\PowerKrabsEtw-1.0.2-alpha> $events[0]

EtwEventId      : 5
EtwTimestamp    : 7/14/2020 8:38:50 PM
EtwProcessId    : 5140
EtwThreadId     : 3684
EtwProviderName : Microsoft-Windows-Kernel-Process
ImageBase       : 140694925934592
ImageSize       : 356352
ProcessID       : 5140
ImageCheckSum   : 0
TimeDateStamp   : 1592823736
DefaultBase     : 140694925934592
ImageName       : \Device\HarddiskVolume2\tools\DemoProcess.exe

```

Bild B.5: PowerKrabsETW (v.0.1)

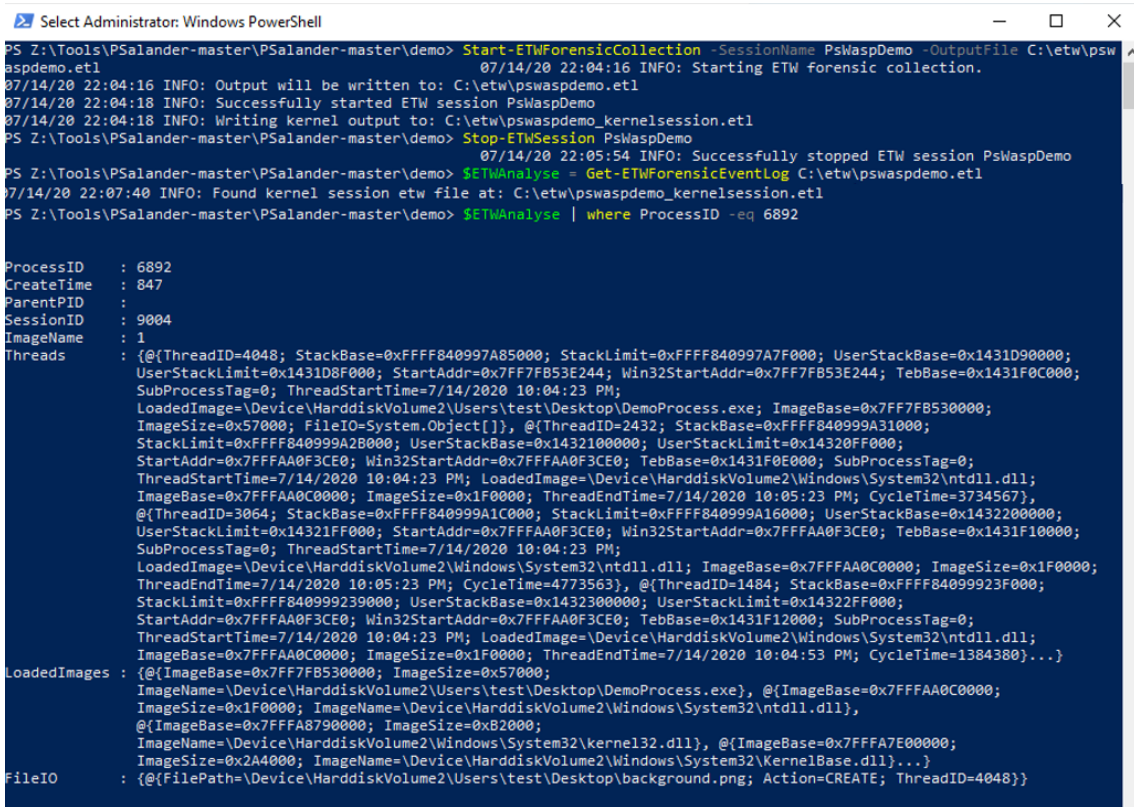


Bild B.6: Analysetool: PSWasp (v.0.0.0.1)

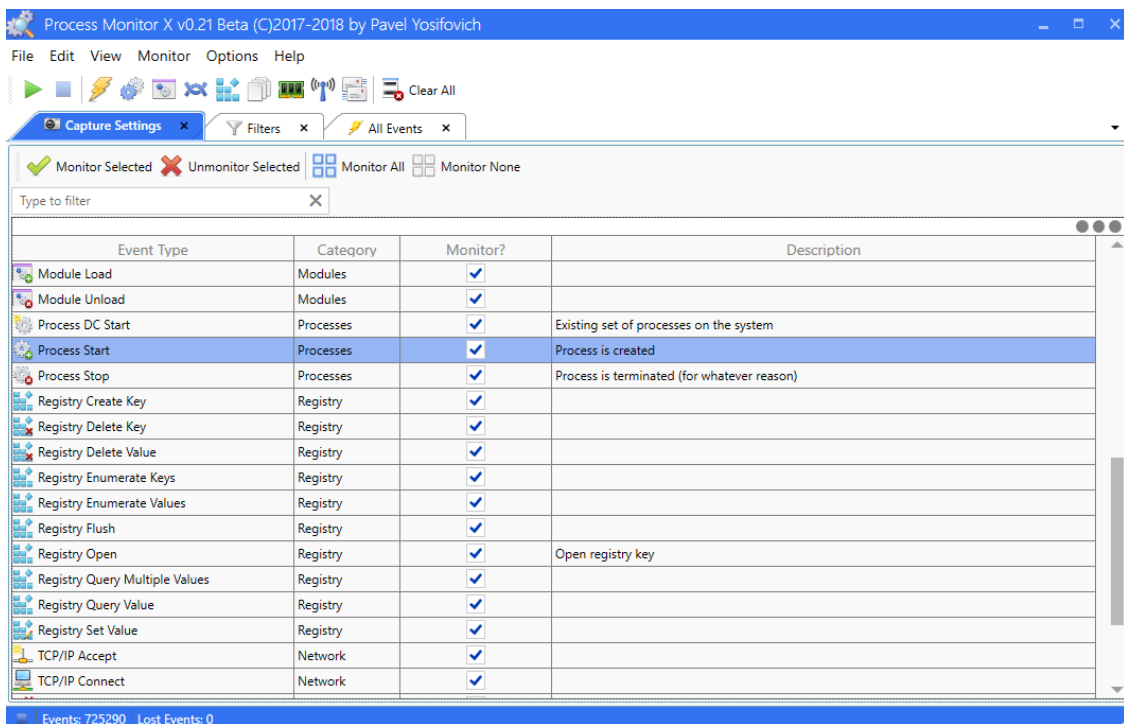


Bild B.7: ProcMonX (0.21 Beta)

C Windows Kernel Trace (NT Kernel Logger)

Tabelle C.1: Zuordnung der Kernel Keywords (ETW-API) zur TraceEvent Library

ETW-API			TraceEvent Library (.NET API)		
Keyword	Wert	Beschreibung ¹	Keyword	Kommentar der Entwickler ²	Events pro Sek. ³
process	0x1	Process creations/deletions	Process	Logs process starts and stops.	
thread	0x2	Thread creations/deletions	Thread	Logs threads starts and stops	
img	0x4	Image load	ImageLoad	Logs native modules loads (LoadLibrary) and unloads	
procntr	0x8	Process counters	ProcessCounters	Logs process performance counters (Vista+ only)	
cswitch	0x10	Context switches	ContextSwitch	log thread context switches (Vista only)	>10K
dpc	0x20	Deferred procedure calls	DeferedProcedureCalls	log deferred procedure calls (an Kernel mechanism for having work done asynchronously) (Vista+ only)	
isr	0x40	Interrupts	Interrupt	log hardware interrupts. (Vista+ only)	
syscall	0x80	System calls	SystemCall	log calls to the OS (Vista+ only)	>100K
disk	0x100	Disk IO	DiskIO	Loads the completion of Physical disk activity.	
file	0x200	File details	DiskFileIO	Logs the mapping of file IDs to actual (kernel) file names.	
diskinit	0x400	Disk IO entry	DiskIOInit	log Disk operations (Vista+ only)	
dispatcher	0x800	Dispatcher operations	Dispatcher	Thread Dispatcher (ReadyThread) (Vista+ only)	>10K
pf	0x1000	Page faults	Memory	Logs all page faults (hard or soft)	>1K
hf	0x2000	Hard page faults	MemoryHardFaults	Logs all page faults that must fetch the data from the disk (hard faults)	
virtualloc	0x4000	Virtual memory allocations	VirtualAlloc	Log Virtual Alloc calls and VirtualFree. (Vista+ Only)	<1K
net	0x10000	Network TCP/IP	NetworkTCPIP	Logs TCP/IP network send and receive events.	
registry	0x20000	Registry details	Registry	Logs activity to the windows registry.	>1K
alpc	0x100000	ALPC	AdvancedLocalProcedureCalls	Logs Advanced Local Procedure call events.	
splitio	0x200000	Split IO	SplitIO	Disk I/O that was split (eg because of mirroring requirements) (Vista+ only)	
driver	0x800000	Driver delays	Driver	Device Driver logging (Vista+ only)	
profile	0x1000000	Sample based profiling	Profile	Sampled based profiling (every msec) (Vista+ only)	1K
fileiocompletion	0x2000000	File IO completion	FileIO	log file FileOperationEnd (has status code) when they complete (even ones that do not actually cause Disk I/O). (Vista+ only)	<1K
fileio	0x4000000	File IO	FileIOInit	log the start of the File I/O operation as well as the end. (Vista+ only)	<1K
-/-	-/-	-/-	VAMap	Log mapping of files into memory (Win8 and above Only)	low vol.

¹ Informationen zu den Event Strukturen der Kernel Provider: [https://msdn.microsoft.com/en-us/library/aa363784\(VS.85\).aspx](https://msdn.microsoft.com/en-us/library/aa363784(VS.85).aspx)

² Kommentierung im Quellcode (KernelTraceEventParser.cs) aus der TraceEvent Library (Microsoft.Diagnostics.Tracing.TraceEvent)

³ Entwicklerangabe

Tabelle C.2: Kernel Keyword-Collections (TraceEvent Library)

Keyword (TraceEvent library)	Zugeordnete Kernel Keywords	Kommentar (siehe TraceEvent Library) ¹
Default	DiskIO, DiskFileIO, DiskIOInit, ImageLoad, MemoryHardFaults, NetworkTCPIP, Process, ProcessCounters, Profile, Thread	Good default kernel flags.
Verbose	Default, ContextSwitch, Dispatcher, FileIO, FileIOInit, Memory, Registry, VirtualAlloc, VAMap	These events are too verbose for normal use, but this give you a quick way of turing on 'interesting' events. This does not include SystemCall because it is 'too verbose'
ThreadTime	Default, ContextSwitch, Dispatcher	Use this if you care about blocked time.
OS	AdvancedLocalProcedureCalls, DeferedProcedureCalls, Driver, Interrupt, SplitIO	You mostly don't care about these unless you are dealing with OS internals.
All	Verbose, ContextSwitch, Dispatcher, FileIO, FileIOInit, Memory, Registry, VirtualAlloc, VAMap , SystemCall, OS	All legal kernel events

¹ Kommentierung im Quellcode (KernelTraceEventParser.cs) aus der TraceEvent Library (Microsoft.Diagnostics.Tracing.TraceEvent)**Tabelle C.3:** Kernel Keyword-Collection: Forensic (ETWProcessTracer)

Keyword (ETWProcessTracer)	Zugeordnete Kernel Keywords (TraceEvent Library)	Entspricht Kernel Keywords (ETW-API)	Kommentar
Forensic	Process, Thread, ImageLoad, NetworkTCPIP, FileIOInit, Registry	process (0x1), thread (0x2), img (0x4), net (0x1000), fileio (0x04000000), registry (0x20000)	Die Auswahl der Kernel-Keywords liefert relevante Events aus den Kernel Trace Providern. Die Anzahl der Events ist vom Prozessverhalten abhängig.

D Forensisch relevante ETW-Provider aus Kap. 6 (PoC)

D.1 Prozesse und Threads

Tabelle D.1: Microsoft-Windows-Kernel-Process (forensisch rel. Events)

EventName	Verhalten	Daten (for. rel.)
ImageLoad	Image eingebunden	ProcessID, ImageName (Pfad)
ImageUnload	Image ausgebinden	ProcessName (Pfad zum Image)
ProcessStart/Start	Prozessstart	ProcessID, ParentPID, Startzeit
ProcessStart/Stop	Prozessende	ProcessID, ParentPID, Startzeit, Endzeit, Exit-Code, HandleCount, CPUCycleCount, ReadOperationCount, WriteOperationCount, ReadTransferKiloBytes, WriteTransferKiloBytes, ImageName (unvollständig)
ThreadStart/Start	Threaderstellung	ProcessID, ThreadID, Stack-Adressen, Win32StartAddr

Tabelle D.2: Microsoft-Windows-Kernel-Prefetch (forensisch rel. Events)

EventName	Verhalten	Daten (for. rel.)
ScenarioDecision	Prefetch-Szenario erstellt	ScenarioName, Typ, NumLaunches (<i>Anzahl d. bisherigen Ausführungen</i>), TimeSinceLastLaunchInS

Tabelle D.3: Windows Kernel: Process / process (forensisch rel. Events)

EventName	Verhalten	Daten (for. rel.)
Process/DCStart	Bereits ausgeführter Prozess	ProcessID, ParentPID, ImageFileName, CommandLine
Process/Start	Prozessstart	ProcessID, ParentPID, ImageFileName, CommandLine
Process/Stop	Prozessende	ProcessID, ParentPID, ImageFileName, CommandLine

Weiterführende Informationen: <https://docs.microsoft.com/en-us/windows/win32/etw/process>

Tabelle D.4: Windows Kernel: ImageLoad / img (forensisch rel. Events)

EventName	Verhalten	Daten (for. rel.)
Image/DCStart	Bereits eingebundenes Image	FileName (Pfad zur ausführbaren Datei / Bibliothek)
Image/Load	Image eingebunden	FileName (Pfad zum Image)
Image/Unload	Image ausgebinden	FileName (Pfad zum Image)

Weiterführende Informationen: <https://docs.microsoft.com/en-us/windows/win32/etw/image>

Tabelle D.5: Microsoft-Windows-DotNETRuntime (forensisch rel. Events)

EventName	Verhalten	Daten (for. rel.)
Exception/Start	Auslösen einer Exception	ExceptionMessage, ExceptionHRESULT,
Method/Inlining-Failed	Inlining-Namespace Fehler	MethodBeingCompiledName, InlineeNamespace, FailReason

D.2 Dateisystem

Tabelle D.6: Microsoft-Windows-Kernel-File (forensisch rel. Events)

EventName	Verhalten	Daten (for. rel.)
Create	File Handle erstellt	FileObject, FileName (Pfad)
CreateNewFile	Neue Datei erstellt	FileObject, FileName (Pfad)
NameCreate	Dateiname erstellt	FileKey, FileName (Pfad)
RenamePath	Umbenennen eines (Datei-)Pfades	FileKey, FilePath

Tabelle D.7: Microsoft-Windows-FileInfoMinifilter (forensisch rel. Events)

EventName	Verhalten	Daten (for. rel.)
fi:FileNameCreate	Dateiname erstellt	Path (Dateipfad)

Tabelle D.8: Microsoft-Windows-Shell-Core (forensisch rel. Events)

EventName	Verhalten	Daten (for. rel.)
CDesktopFolder_ParseDisplayName/Start	Parsing des Display-Namens	Name (Pfad zum Image)
Shell32_CopyEngine_FileOperation_Info	Kopiervorgang	pszSource (Quelldatei), pszDest (Zieldatei)
ShellTask_ExecAssoc_ZoneCheckFile/Stop	Zonen-Check	Name (Pfad zum Image)

Tabelle D.9: Windows Kernel: FileIO / fileiocompletion (forensisch rel. Events)

EventName	Verhalten	Daten (for. rel.)
FileIO/Cleanup	File Handle geschlossen	FileName (Pfad)
FileIO/Close	File schließen	FileName (Pfad)
FileIO/Create	File Handle erstellt	CreateOptions, ShareAccess, FileName (Pfad)
FileIO/Delete	Datei löschen	FileName (Pfad)
FileIO/DirEnum	Verzeichnisaufruf	DirectoryName (Pfad)
FileIO/FSCControl	FSCTL-Befehl ausgelöst	FileName (Pfad)
FileIO/MapFile	Mapping einer Datei	FileName(Pfad)
FileIO/QueryInfo	Dateiinformation abfragen	FileName (Pfad)
FileIO/Read	Datei lesen	FileName (Pfad), Offset, IOSize
FileIO/UnMapFile	Unmapping einer Datei	FileName(Pfad)
FileIO/Write	Datei schreiben	FileName (Pfad), Offset, IOSize

Weiterführende Informationen: <https://docs.microsoft.com/en-us/windows/win32/etw/fileio>

D.3 Windows-Registry

Tabelle D.10: Microsoft-Windows-Kernel-Registry (forensisch rel. Events)

EventName	Verhalten	Daten (for. rel.)
EventID(1)	CreateKey	RelativeName (Registry-Key, Pfad), KeyObject (HEX)
EventID(2)	OpenKey	RelativeName (Registry-Key, Pfad), KeyObject (HEX)
EventID(5)	SetValueKey	ValueName, KeyObject (HEX)
EventID(7)	QueryValueKey	ValueName, KeyObject (HEX)

Tabelle D.11: Windows Kernel: Registry (forensisch rel. Events)

EventName	Verhalten	Daten (for. rel.)
Registry/KCB-Create	Create Key Event	KeyName (Registry-Key, Pfad)
Registry/Open	OpenKey	KeyName (Registry-Key, Pfad)

Weiterführende Informationen: <https://docs.microsoft.com/en-us/windows/win32/etw/registry>

D.4 Netzwerk und Kommunikation

Tabelle D.12: Microsoft-Windows-CAPI2 (forensisch rel. Events)

EventName	Verhalten	Daten (for. rel.)
BuildChain/Stop	CertGetCertificateChain abgeschlossen (Wincrypt.h). Zusammenfassung der Zertifikatskette.	Auflistung der gesamten Zertifikatskette mit Revocation-Status, u.a. Certificate fileRef, subjectName, IssuerCertificate fileRef, RevocationStatus, AdditionalParameters currentTime, error, reason, actualFreshnessTime, OSCP-URL, issuerName, Usage, PKI-Informationen und Zertifikatsalgorithmen
RetrieveObject-fromNetwork/Stop	CryptRetrieveObjectByUrlA abgeschlossen (Wincrypt.h). Abruf PKI-Objekt von einer URL.	Parameter des abgerufenen Objekts (Parameter: pszObjectOid, dwRetrievalFlags), z.B. OCSP-URL, HTTP-Header
VerifyChainPolicy	CertVerifyCertificateChainPolicy abgeschlossen (Wincrypt.h).	Certificate fileRef, subjectName, CertificateChain chainRef, authType, serverName,
VerifyRevocation/-Stop	CertVerifyRevocation abgeschlossen (Wincrypt.h). Status: Zertifikatssperrlistenprüfung.	Informationen und Ergebnis der Zertifikatssperrprüfung. Informationen zum Überprüften Zertifikat: Certificate fileRef, subjectName, IssuerCertificate fileRef, RevocationStatus, AdditionalParameters currentTime, error, reason, actualFreshnessTime, OSCP-URL, issuerName
VerifyTrust/Stop	WinVerifyTrust abgeschlossen (Wintrust.h)	CertificateInfo displayName, RevocationCheck value,
X509Objects	Zusammenfassung der X509-Objekte der vollständigen Zertifikatskette	X509-Felder (issuerName, OU, PN, Algorithmen, KeyUsage, DNS-Namen, Gültigkeit, Seriennummern)

Tabelle D.13: Microsoft-Windows-DNS-Client (forensisch rel. Events)

EventName	Verhalten	Daten (for. rel.)
EventID(3006)	DNS-Abfrage gestartet	QueryName
EventID(3007)	DnsQueryEx aufgerufen (Windns.h)	QueryName
EventID(3008)	DNS-Abfrage abgeschlossen	QueryName, Aufgelöste IP-Adresse(n)

Tabelle D.14: Microsoft-Windows-MPS-CLNT (forensisch rel. Events)

EventName	Verhalten	Daten (for. rel.)
MPS_CLNT_API_SetFirewallRule/Start	Firewallregel erstellen	nur EventName
MPS_CLNT_API_SetFirewallRule/Stop	Firewallregel erstellt	nur EventName

Tabelle D.15: Microsoft-Windows-RPC (forensisch rel. Events)

EventName	Verhalten	Daten (for. rel.)
RpcClientCall/Start	Client RPC Aufruf gestartet	Protocol (z.B. LRPC), NetworkAddress, Endpoint, AuthenticationLevel, AuthenticationService, ImpersonationLevel

Tabelle D.16: Microsoft-Windows-TCPIP (forensisch rel. Events)

EventName	Verhalten	Daten (for. rel.)
TcpipRouteLookup	IP-Routenaufösung mit API Call: IppFindPath	DestAddr (Zieladresse), DestinationAddress
TcpConnectTcb-Proceeding	TCP-Verbindung wird hergestellt.	LocalAddress (IP/Port), RemoteAddress (IP/Port)
TcpRequest-Connect	TCP-Verbindungsanfrage	lokale IP-Adresse/Port, entfernte IP-Adresse/Port
TcpConnectRe-transmit	Wiederholen der TCP-Verbindungsanfrage	lokale IP-Adresse/Port, entfernte IP-Adresse/Port

Tabelle D.17: Microsoft-Windows-URLMon (forensisch rel. Events)

EventName	Verhalten	Daten (for. rel.)
URLMON_CInet_Start	Binding der URL zur Ressource	URL
URLMON_Queue_Msg	Zustandsänderung einer URL in der Queue	URL, Msg (z.B. PROXYDETECTING, BEGINDOWNLOADDATA)
URLMON_Process_Queued_Msg	Verarbeitung einer URL in der Queue	URL, Msg (z.B. MIMETYPEAVAILABLE, BEGINDOWNLOADDATA)
URLMON_CInet_Read	Download der Ressource	URL, Bytes, Msg (z.B. ENDDOWNLOADDATA)

Tabelle D.18: Microsoft-Windows-WebIO (forensisch rel. Events)

EventName	Verhalten	Daten (for. rel.)
RequestCreate	WebCreateHttpRequest abgeschlossen	Methode (GET), URL-Pfad
ConnectionName-Resolution/Start	Adressauflösung gestartet	ResolveName (FQDN)
ConnectionName-Resolution/Stop	Adressauflösung abgeschlossen	ResolveName (FQDN), AddressData (IP-Adresse in HEX),

Tabelle D.19: Microsoft-Windows-WinINet (forensisch rel. Events)

EventName	Verhalten	Daten (for. rel.)
Wininet_Connect/Stop	Verbindung beendet	LocalAddress (IP/Port), RemoteAddress (IP/Port)
WININET_CONNECT_HANDLE_CREATED	InternetConnect (Wininet.h), Handlerzeugung	Server (IP, Port), Service (z.B. HTTP), ServerName, ServerPort
WININET_DNS_QUERY/Start	DNS-Auflösungsanfrage gesendet	Hostname
WININET_DNS_QUERY/Stop	DNS-Auflösungsanfrage beendet	Hostname, AddressList (IP)
Wininet_Getaddrinfo/Start	DNS-Auflösungsanfrage gesendet	Hostname
WININET_HTTP_REQUEST_HANDLE_CREATED	HttpOpenRequest (Wininet.h), Handlerzeugung	Verb (HTTP-Methode), ObjectName, Version, Referrer
WININET_HTTP_REQUEST_HANDLE_CREATED	Details zum Request-Handle	Server (IP, Port), Service (z.B. HTTPS)
WININET_HTTP_RESPONSE_BODY_RECEIVED	HTTP Response Body empfangen	nur Information
WININET_HTTPS_SERVER_CERT_VALIDATED	Serverzertifikatsvalidierung	CertHash (HEX)
WININET_REQUEST_HEADER	HTTP Request Headers	Header (Methode, Object, Host, ContentLength)
WININET_REQUEST_HEADER_OPTIONAL	HTTP Request Headers OptionalData	Content-Disposition, Content-Type, application/ octet-stream
Wininet_ResolveHost/Start	Auflösen des Hostnames gestartet	Hostname
WININET_RESPONSE_HEADER	HTTP Response Headers	Header (Methode, Object, Host, ContentLength)
Wininet_SendRequest/Start	Start des Request-Sendevorgangs	AddressName (URL)
Wininet_SendRequest/Start	Stop des Request-Sendevorgangs	AddressName (URL), StatusLine
WININET_TCP_CONNECTION/Start	Start der TCP-Verbindung	ServerName, LocalPort
Wininet_UsageLogRequest	Zusammenfassung	Requested URL, RequestHeaders, Content-Length, ResponseHeaders, Content-Length, Status, URL, Verb

Tabelle D.20: Microsoft-Windows-WinINet-Capture (forensisch rel. Events)

EventName	Verhalten	Daten (for. rel.)
EventID(2001)	Aufgezeichneter Buffer: Request-Header	Payload-Data (HEX)
EventID(2002)	Aufgezeichneter Buffer: Request-Payload	Payload-Data (HEX)
EventID(2003)	Aufgezeichneter Buffer: Response-Header	Payload-Data (HEX)
EventID(2004)	Aufgezeichneter Buffer: Response-Payload	Payload-Data (HEX)

Tabelle D.21: Microsoft-Windows-Winsock-AFD (forensisch rel. Events)

EventName	Verhalten	Daten (for. rel.)
AfdAbort/Aborted	Socket-abort()	Ursache
AfdBindWithAddress/Open	Socket-bind()	Lokale-Adresse (0.0.0.0:Port)
AfdClose/Closed	Socket-closesocket()	nur Information
AfdConnectWithAddress/Bound	Socket-connect()	Remote-Adresse (IP/Port)
AfdReceive/Connected	Socket-recv()	BufferLength, Seq.
AfdSend/Connected	Socket-send()	BufferLength, Seq.

Weiterführende Informationen: <https://docs.microsoft.com/en-us/windows/win32/winsock/winsock-tracing-events>

Tabelle D.22: Microsoft-Windows-Winsock-NameResolution (forensisch rel. Events)

EventName	Verhalten	Daten (for. rel.)
WinsockGai	GetAddrInfoExW aufgerufen (Ws2tcpip.h)	QueryName
WinsockGai	GetAddrInfoExW abgeschlossen (Ws2tcpip.h)	QueryName, Aufg. IP-Adresse(n)
WinsockGai	NSPLookupServiceNext abgeschlossen	Aufgelöste IP-Adresse

Tabelle D.23: Windows Kernel: TcpIp (forensisch rel. Events)

EventName	Verhalten	Daten (for. rel.)
TcpIp/Connect	TCP-Verbindungsaufbau	daddr, saddr, dport, sport, mss, rcvwin, size
TcpIp/Disconnect	TCP-Verbindungsabbau	daddr, saddr, dport, sport, seqnum, size
TcpIp/Recv	TCP-Daten empfangen	daddr, saddr, dport, sport, seqnum, size
TcpIp/Send	TCP-Daten gesendet	daddr, saddr, dport, sport, starttime, enddtime, seqnum, size

Weiterführende Informationen: <https://docs.microsoft.com/en-us/windows/win32/etw/tcpip>

D.5 Skripting

Tabelle D.24: Microsoft-Windows-PowerShell (forensisch rel. Events)

EventName	Verhalten	Daten (for. rel.)
ExecuteaRemoteCommand/ Oncreatecalls	Skriptblockerstellung	PowerShell-Skriptblock (ScriptBlockText), ScriptBlock ID
StartingCommand/ Tobeusedwhenoperationis-justexecutingamethod	Skriptausführung	PowerShell-Befehlsaufruf oder Skriptaufufruf, Command Name, Command Type, Command Path, Script Name, User
StartingEngine/ Tobeusedwhenoperationis-justexecutingamethod	Engine-Zustandswechsel: (z.B. Verfügbar)	PowerShell-Aufruf mit Parameter (Host Application), Command Name, Command Type, Command Path, Script Name, User
StartingProvider/ Tobeusedwhenoperationis-justexecutingamethod	Provider-Zustandswechsel (z.B. Gestartet)	PowerShell-Aufruf mit Parameter (Host Application), Command Name, Command Type, Command Path, Script Name, User

E Konfigurationsparameter der Analyseumgebung

E.1 Virtualisierungsumgebung

Host-System:

- HPM Live Version vom 12.05.2020 (Ubuntu 20.04)
 - Image: *20200512_2004_hpm.iso*
 - SHA1: *4F70F0DE75A2E11126B12F34F4133F10B7FC6EA4*

Virtualisierungsplattform:

- Oracle VirtualBox Version 6.1.6

Konfiguration der VM des Analysesystems (Auszug):

```
<VirtualBox xmlns="http://www.virtualbox.org/" version="1.15-linux">
<Machine uuid="{c855b9cc-2e33-4aea-841e-b3b9e5e770a5}" name="Win10MW" OSType="Windows10_64" currentSnapshot="{1
d5d65f8-6dff-4649-8a14-a1de0fe9ffd9}" snapshotFolder="Snapshots" lastStateChange="2020-07-30T17:50:38Z">
<MediaRegistry>
<HardDisks>
<HardDisk uuid="{5ee919cf-055a-413c-a5ce-ce66436515d6}" location="Win10MW.vdi" format="VDI" type="Normal">
</HardDisk>
</HardDisk>
[...]
```

```
<Snapshot uuid="{877d0d46-0fae-4f83-b23a-d647ddadfbab}" name="Snapshot 1" timeStamp="2020-07-06T10:54:30Z">
  <Hardware>
    <CPU count="2">
      <PAE enabled="true"/>
      <LongMode enabled="true"/>
      <HardwareVirtExLargePages enabled="false"/>
    </CPU>
    <Memory RAMSize="4096"/>
    <HID Pointing="USBTablet"/>
    <Boot>
      <Order position="1" device="DVD"/>
      <Order position="2" device="HardDisk"/>
      <Order position="3" device="None"/>
      <Order position="4" device="None"/>
    </Boot>
    <Display controller="VBoxSVGA" VRAMSize="128"/>
    <VideoCapture screens="1" file="." fps="25"/>
    <RemoteDisplay enabled="false"/>
    <BIOS>
      <IOAPIC enabled="true"/>
      <SmbiosUuidLittleEndian enabled="true"/>
    </BIOS>
    <USB>
      <Controllers>
        <Controller name="XHCI" type="XHCI"/>
      </Controllers>
    </USB>
    <Network>
      <Adapter slot="0" enabled="true" MACAddress="080027F1DF79" cable="true" type="82540EM">
        <NAT/>
      </Adapter>
    </Network>
```

```

<AudioAdapter controller="HDA" driver="ALSA" enabled="true" enabledIn="false"/>
<Clipboard/>
<GuestProperties>
  <GuestProperty name="/VirtualBox/HostInfo/GUI/LanguageID" value="en_US" timestamp="
    1596114800660427000" flags="RDONLYGUEST"/>
  <GuestProperty name="/VirtualBox/HostInfo/VBoxRev" value="137129" timestamp="1596114800632028002"
    flags="TRANSIENT, RDONLYGUEST"/>
  <GuestProperty name="/VirtualBox/HostInfo/VBoxVer" value="6.1.6" timestamp="1596114800632028000"
    flags="TRANSIENT, RDONLYGUEST"/>
  <GuestProperty name="/VirtualBox/HostInfo/VBoxVerExt" value="6.1.6" timestamp="1596114800632028001"
    flags="TRANSIENT, RDONLYGUEST"/>
  <GuestProperty name="/VirtualBox/VMInfo/ResetCounter" value="2" timestamp="1596116321164392000" flags
    ="TRANSIENT, RDONLYGUEST"/>
  <GuestProperty name="/VirtualBox/VMInfo/ResumeCounter" value="1" timestamp="1596114800632050000"
    flags="TRANSIENT, RDONLYGUEST"/>
</GuestProperties>
</Hardware>
<StorageControllers>
  <StorageController name="SATA" type="AHCI" PortCount="3" useHostIOPCache="false" Bootable="true"
    IDE0MasterEmulationPort="0" IDE0SlaveEmulationPort="1" IDE1MasterEmulationPort="2"
    IDE1SlaveEmulationPort="3">
    <AttachedDevice type="HardDisk" hotpluggable="false" port="0" device="0">
      <Image uuid="{6490cb0a-6378-4cc6-917e-c7f6c2312ca5}"/>
    </AttachedDevice>
    <AttachedDevice passthrough="false" type="DVD" hotpluggable="false" port="1" device="0"/>
  </StorageController>
</StorageControllers>
</Snapshot>
</Machine>
</VirtualBox>

```

Listing E.1: VirtualBox VM-Konfiguration des Analysesystems

E.2 Analysesystem (VM): Windows 10 Version 1909 (Education)

Installationsmedium:

- ISO: *en_windows_10_consumer_editions_version_1909_x64_dvd_be09950e.iso*
- SHA1: *9DC9CD4D956C51FDAA9ECD0F86FDF0A1E35526D9*
- Sprache: *English (US)*
- Version: *1909 (x64)*
- Build: *10.0.18363.836*
- Edition: *Education*

Installationsparameter:

- Language to install: *English (United States)*
- Time and currency format: *German (Germany)*
- Keyboard or input method: *German*
- Operating system: *Windows 10 Education*
- Type of installation: *Custom: Install Windows only (advanced)*
- Disk Drive: *50 GB*
- Region: *Germany*
- Keyboard layout: *German (Germany)*
- Internet Connection: *"I don't have Internet" (Continue with limited setup)*
- Username: *JohnDoe*
- Privacy Settings: *All declined*

Anpassungen in Windows 10:

1. User Account Control (UAC): *Never notify*

2. Windows Defender:

GPO-Path: Computer->Administrative Settings->Windows Components->Windows Defender

- Turn off Windows Defender Antivirus: *Enabled*
- Turn off routine remediation: *Enabled*
- Configure detection for potentially unwanted applications: *Disabled*
- Allow antimalware service to remain running always: *Disabled*

3. Windows Update:

GPO-Path: Computer->Administrative Settings->Windows Components->Windows Update)

- Configure Automatic Updates: *Enabled (2: Notify for download and auto install)*
- Do not include drivers with Windows Updates: *Enabled*

Netzwerkkonfiguration:

- Computername: *DESKTOP-01EI200* (PoC I/II), *DESKTOP-01E0000* (PoC III)
- Physical Address: *08:00:27:F1:DF:79*
- IPv4 Address: *10.0.2.15*
- Subnet Mask: *255.255.255.0*
- Default Gateway: *10.0.2.2*
- DNS-Server: *10.0.2.3*

Installierte Programme und Windows Features (abweichend vom Standard):

- 7zip 20.00 alpha (x64)
- Google Chrome (84.0.4147.105)
- Microsoft Office Home and Student 2013 (de-de, 15.0.5249.1001)
- Microsoft Visual C++ 2015-2019 Redistributable (x64)
- Microsoft Visual C++ 2015-2019 Redistributable (x86)
- Mozilla Firefox 78.0.1 (x64 en-US)
- Notepad++ (32-bit x86)
- .NET Framework 3.5 (includes .NET 2.0 and 3.0)

Installierte Windows-Updates:

- 2020-07 Cumulative Update Preview for .NET Framework 3.5 and 4.8 for Windows 10 Version 1909 x64 (KB4562900)
- 2020-07 Cumulative Update for Windows 10 Version 1909 for x64-based Systems (KB4565483)
- 2020-06 Cumulative Update for Windows 10 Version 1909 for x64-based Systems (KB4560960)
- 2020-06 Security Update for Adobe Flash Player for Windows 10 Version 1909 for x64-based Systems (KB4561600)
- 2020-06 Cumulative Update for .NET Framework 3.5 and 4.8 for Windows 10 Version 1909 x64 (KB4552931)

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die hier vorliegende Arbeit selbstständig, ohne unerlaubte fremde Hilfe und nur unter Verwendung der aufgeführten Hilfsmittel angefertigt habe.

Ort, Datum

Martin Reuter

Thesen

1. Event Tracing for Windows (ETW) ist ein Framework zur Erfassung und Aufzeichnung von Ereignissen und der Ablaufverfolgung von (System-)Komponenten, Treibern und Anwendungen in Windows-Betriebssystemen (ab Windows 2000).
2. Im Vergleich zur Ereignisprotokollierung lassen sich mit ETW dynamische Abläufe und Aktivitäten aufzeichnen und detailliert nachverfolgen.
3. Für Softwareherausgebende, -entwickelnde und Programmierende ist der Einsatz von ETW zum Debugging, zur Performanz- und Leistungsanalyse sowie zur Erfassung von Telemetriedaten von Software eine gängige Praxis.
4. Im Umfeld der IT-Sicherheit und Forensik erlangt ETW zunehmend größere Bekanntheit und Relevanz.
5. Der Einsatz von ETW zur Prozessverhaltensanalyse in Windows 10 ist als Kernkomponente Out-of-the-box (mit Administratorrechten) möglich und erfordert keine zusätzlichen Komponenten.
6. Als umfangreiche Informations- und Datenquelle dienen zahlreiche ETW-Provider im User-Mode als auch im Kernel-Mode.
7. In vorausgehenden Forschungsarbeiten und -projekten wurden bereits forensisch relevante ETW-Provider als Informations- und Datenquelle für die Live- und Post-Mortem Forensik identifiziert.
8. ETW ermöglicht eine Unterstützung forensischer Analysen von Prozessverhalten in Windows 10, insbesondere in der Malware-Forensik.
9. (Weitere) Forensisch relevante ETW-Provider und Events lassen sich durch Aufzeichnung und Auswertung bestimmter Anwendungsfälle nach dem Maximalprinzip bestimmen.
10. Erkenntnisse aus Prozessverhaltensanalysen mit ETW stellen eine Grundlage für die Prävention und Detektion von Angriffen und Bedrohungen dar und können zur Reaktion auf (erfolgreiche) Angriffe herangezogen werden.