Federal Office
for Information Security

TMFW

Version: 1.0

Federal Office for Information Security
Post Box 20 03 63
D-53133 Bonn
Phone: +49 22899 9582-0
E-Mail: bsi@bsi.bund.de
Internet: https://www.bsi.bund.de

Table of Contents

Figures

Tables

Code Blocks

1 Introduction

1.1 Zusammenfassung

Dieses Dokument stellt das Ergebnis von Arbeitspaket TMFW des Projekts „SiSyPHuS Win10: Studie zu Systemaufbau, Protokollierung, Härtung und Sicherheitsfunktionen in Windows 10“ dar. Das Projekt wird durch die Firma ERNW Enno Rey Netzwerke GmbH im Auftrag des Bundesamts für Sicherheit in der Informationstechnik (BSI) durchgeführt.

Ziel dieses Arbeitspakets ist die Bereitstellung einer technischen Lösung zur Überwachung der im Arbeitspaket 4 analysierten Telemetrie-Komponente während der Ausführung (ERNW_WP4). Hierzu wurde eine Anwendung entwickelt, die eine detaillierte Erfassung des System- und Anwendungsverhaltens sowie der Ressourcennutzung für Forschungszwecke ermöglicht – genannt System Activity Monitor. Die Anwendung verwendet die vom Betriebssystem bereitgestellte aufwandsarme Hochgeschwindigkeits-„Event Tracing for Windows“-Infrastruktur, um Verhaltens- und Ressourcennutzungsdaten zu sammeln. Sie erweitert „Event Tracing for Windows“ und ermöglicht detaillierte Aufzeichnungen des System- und Anwendungsverhaltens sowie der Ressourcennutzung basierend auf sogenannten Aufzeichnungsprofilen (Recording Profiles). Damit ermöglicht die Anwendung eine umfassende Informationssammlung für Forschung und Systemanalyse, indem es während der Ausführung generierte Systemaktivitätsdaten erfasst.

1.2 Executive Summary

This document implements the work plan outlined in Work Package TMFW of the project "SiSyPHuS Win10: Studie zu Systemaufbau, Protokollierung, Härtung und Sicherheitsfunktionen in Windows 10“ (orig., ger.). The project is contracted by the German Federal Office for Information Security (orig., ger., Bundesamt für Sicherheit in der Informationstechnik - BSI).

The objective of this work package is to provide a technical solution for monitoring the Windows telemetry component analyzed in work package 4 during execution (ERNW_WP4). Therefore, a research tool was developed that enables detailed recording of system and application behavior and resource usage – named System Activity Monitor. The tool uses the extremely low overhead and high speed Event Tracing for Windows infrastructure to collect behavior and resource usage data. It extends Event Tracing for Windows and enables detailed recordings of system and application behavior and resource usage based on recording profiles, which makes it an extensive information source for research and system analysis by capturing system activity data generated during execution.

2 Concept and Terms

2.1 Recording Activities

In the context of the operating system, recording means that activities that occur in the operating system and application components are recorded in the form of events (also referred to as records in this work). Some activities are, for example, reading or writing files and starting and stopping processes.

Recording activities is relevant from several perspectives, such as:

- IT security: Recorded activities in a certain environment can be compared with activities typical for the environment in order to identify atypical, possibly malicious activities. This is relevant in order to recognize or understand attacks that are in progress or that have already occurred. If an environment is known to be compromised, analyzing the recorded activities can help determine how the environment was compromised and what countermeasures should be applied.
- Troubleshooting: Analyzing recorded activities of system and application behavior in scenarios where they do not work as expected can help identify the root cause (for example, incorrect implementations or configurations) and in the end correct the problem.
- Component Profiling: A component is the implementation of a single functionality of the Windows operating system, which consists of single or multiple parts of this system. A component profile is the detailed recording and analysis of the behavior and resource usage of a component in a specific scenario. For example, how often was a network connection established by a component and how much data was sent and received. Such a profile can be used in various analyses scenarios, such as: in the context of a statistical analysis of a component, in the case of a security incident, or malware analysis.

The amount of information provided by recorded activities (i.e., record data) is proportional to the number of recorded activities. Recording data on an operating system instance can be a problem if the recording frequency is too high in relation to the central processing unit (CPU) performance, main memory or disk space available to the operating system. This is particularly problematic in scenarios in which the operating system is used productively and must always maintain sufficient CPU performance, main memory or hard disk capacity. This problem is usually resolved by combining different processing and retention approaches. These include: i) deleting records after a certain number of records has been recorded, ii) moving records to an external resource (e.g., network drive or SIEM), and/or iii) distribution of tasks to different systems. How exactly the applied approach is implemented in a given environment depends on the environment in which it is implemented.

2.2 Event Tracing for Windows

Record data in Windows can be created using the Windows built-in recording mechanism Event Tracing for Windows (ETW). ETW is deeply integrated into the operating system, making it an extensive source of system activity data. Most executables that implement operating system and application components come with ETW providers. The ETW application programming interface (API), which is part of Windows, is the central interface to ETW functionalities. This API allows Windows users and third-party software vendors to use ETW in a standardized manner to create new record data and / or access previously generated record data. The architecture of ETW consists of several components: ETW providers, ETW sessions, ETW consumers and ETW controllers.

ETW providers: An ETW provider is a software entity that records activities i.e., produces and delivers record data to ETW sessions. An ETW provider is implemented and declared using the ETW API (see [ERNW WP2], Section 2.5.1), as part of code-instrumented user- or kernel-land resources. Each ETW provider can be uniquely identified by a global unique identifier (GUID) (128-bit number e.g., `00112233-`

`4455-6677-8899-AABBCCDDEEFF`). In addition to a GUID, an ETW providers can also be assigned a name (e.g., ETW provider `Microsoft-Windows-PowerShell`).

In terms of how ETW providers are implemented in code, there are the following types of providers:[1] Managed Object Format (MOF) providers, Windows software trace preprocessor (WPP) providers, Manifest-based providers, and TraceLogging providers.[2] Among other things, providers of these types differ in terms of what functions of the ETW API are used for provider registering, and what information is required for a provider to be registered.

ETW sessions: An ETW session is a software entity implemented in the Windows kernel that receives record data from ETW providers associated with it. ETW has been designed with performance in mind; ETW does not perform expensive logging operations, such as synchronous file flushing using system calls. This results in frequent context switches. Instead, ETW providers delegate data to the Windows kernel, which writes the data to designated memory buffers and asynchronously flushes the data to ETW consumers in real-time, or to files.

ETW consumers: An ETW consumer is software entity that receives and processes (e.g., rendering and viewing recorded data) the data supplied by ETW sessions, such as the Windows Event Viewer utility.

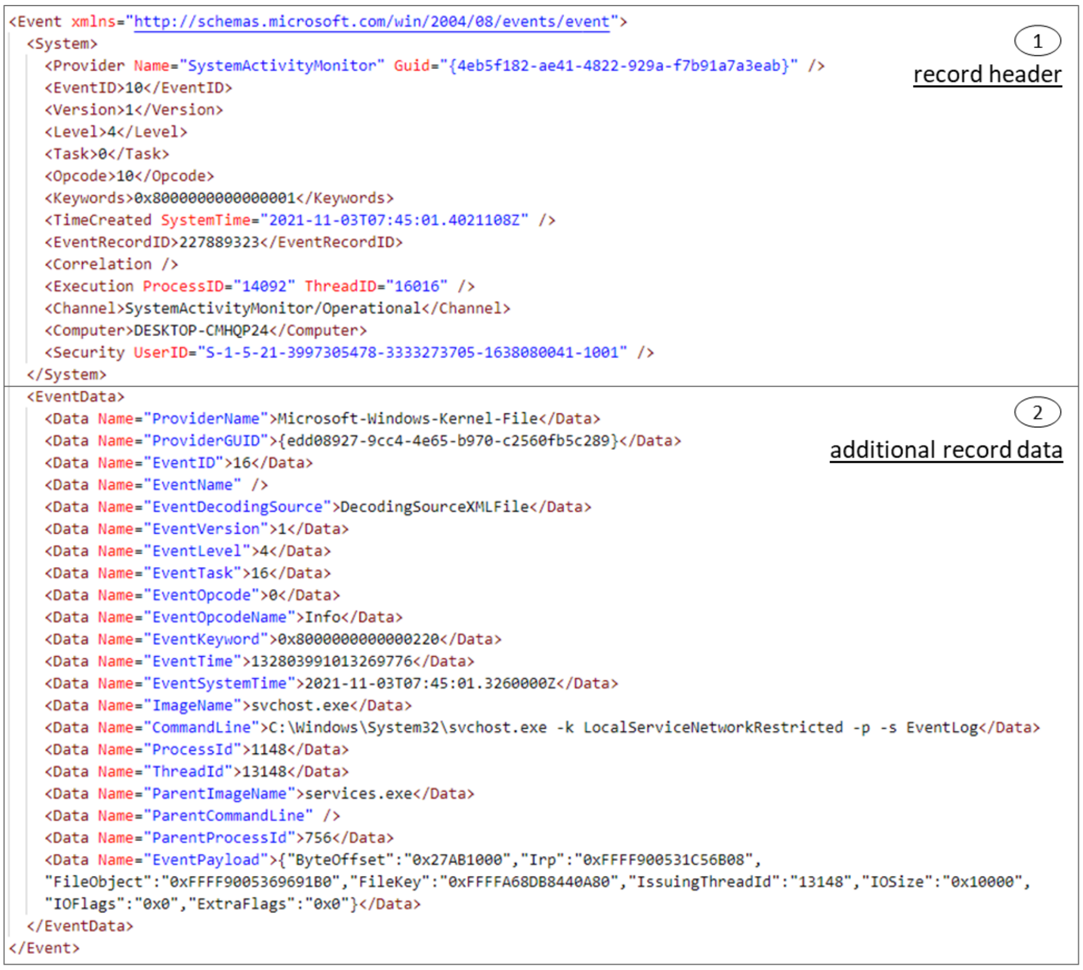
ETW controllers: The management of ETW sessions (e.g., activation and deactivation) and the association of ETW providers to ETW sessions is done by special-purpose applications, referred to as ETW controllers. An example ETW controller (outside of the core operating system, which is directly using the ETW-API) is the xperf utility, distributed as part of the Windows Performance Toolkit.[3]

2.3 Record Data

Recorded activities i.e., the records generated by ETW, have a standard format - each record consists of a record header and additional record data. The record header is in a standard format and contains information about the record, such as: record identifier (ID) - an ID number used to identify the record, the time the record was created, and the process ID of the process that produces the record (msft_ehs, 2020). The format and content of the additional record data does not have a standard format and is defined by the ETW provider that produces and delivers the data.

Most records that can be recorded by a Windows instance have information on the format of the additional record data (event format information). This enables the correct display and interpretation of these records. For example, the Windows Event Viewer utility uses record format information to display records created by ETW providers in Extensible Markup Language (XML) format. Figure 1 shows a record displayed by the Windows Event Viewer in XML format. Information about the record (i.e., the record header) is typically stored under the XML node `<System>` (see Figure 1, label 1), and the additional record data is stored under the XML node `<EventData>` or `<Data>`(see Figure 1, label 2), often as values of XML attributes.

[1]https://msdn.microsoft.com/en-us/library/windows/desktop/aa364161(v=vs.85).aspx [Retrieved:12.08.2021]
[2]https://msdn.microsoft.com/en-us/library/windows/desktop/aa363668(v=vs.85).aspx#providers [Retrieved:12.08.2021]
[3]https://docs.microsoft.com/en-us/windows-hardware/test/wpt/ [Retrieved:12.08.2021]



```xml
<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
  <System>
    <Provider Name="SystemActivityMonitor" Guid="{4eb5f182-ae41-4822-929a-f7b91a7a3eab}" />
    <EventID>10</EventID>
    <Version>1</Version>
    <Level>4</Level>
    <Task>0</Task>
    <Opcode>10</Opcode>
    <Keywords>0x8000000000000001</Keywords>
    <TimeCreated SystemTime="2021-11-03T07:45:01.4021108Z" />
    <EventRecordID>227889323</EventRecordID>
    <Correlation />
    <Execution ProcessID="14092" ThreadID="16016" />
    <Channel>SystemActivityMonitor/Operational</Channel>
    <Computer>DESKTOP-CMHQP24</Computer>
    <Security UserID="S-1-5-21-3997305478-3333273705-1638080041-1001" />
  </System>
  <EventData>
    <Data Name="ProviderName">Microsoft-Windows-Kernel-File</Data>
    <Data Name="ProviderGUID">{edd08927-9cc4-4e65-b970-c2560fb5c289}</Data>
    <Data Name="EventID">16</Data>
    <Data Name="EventName" />
    <Data Name="EventDecodingSource">DecodingSourceXMLFile</Data>
    <Data Name="EventVersion">1</Data>
    <Data Name="EventLevel">4</Data>
    <Data Name="EventTask">16</Data>
    <Data Name="EventOpcode">0</Data>
    <Data Name="EventOpcodeName">Info</Data>
    <Data Name="EventKeyword">0x8000000000000220</Data>
    <Data Name="EventTime">132803991013269776</Data>
    <Data Name="EventSystemTime">2021-11-03T07:45:01.3260000Z</Data>
    <Data Name="ImageName">svchost.exe</Data>
    <Data Name="CommandLine">C:\Windows\System32\svchost.exe -k LocalServiceNetworkRestricted -p -s EventLog</Data>
    <Data Name="ProcessId">1148</Data>
    <Data Name="ThreadId">13148</Data>
    <Data Name="ParentImageName">services.exe</Data>
    <Data Name="ParentCommandLine" />
    <Data Name="ParentProcessId">756</Data>
    <Data Name="EventPayload">{"ByteOffset":"0x27AB1000","Irp":"0xFFFF900531C56B08",
    "FileObject":"0xFFFF9005369691B0","FileKey":"0xFFFFA68DB8440A80","IssuingThreadId":"13148","IOSize":"0x10000",
    "IOFlags":"0x0","ExtraFlags":"0x0"}</Data>
  </EventData>
</Event>
```

Figure 1: Record displayed by the Windows Event Viewer

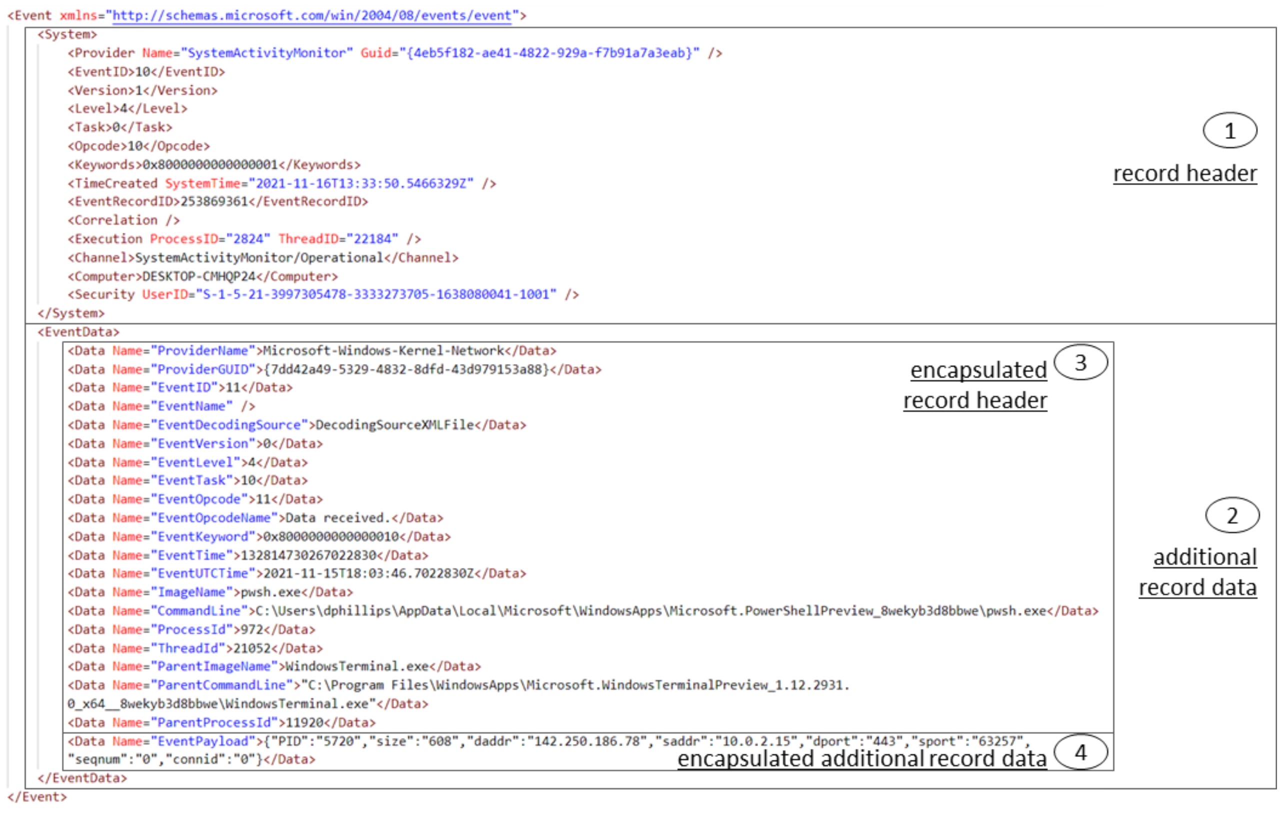
3 System Activity Monitor

System Activity Monitor (SAM) is a research tool that enables detailed recording of system and application behavior and resource usage. SAM uses the extremely low overhead and high speed ETW infrastructure to collect behavior and resource usage data. The decision to use ETW as a basis is mainly based on the fact, that ETW is deeply integrated into the operating system (i.e., it is an integral part of the operating system) which is equivalent to being integrated into every application that runs under Windows. SAM extends ETW and enables detailed recordings based on recording profiles (see section 3.3), which makes it an extensive information source for research and system analysis by capturing system activity data generated during execution.

SAM recording profiles control all aspects of a recording session. A recording profile is a file in XML format that has a `.samp` extension. A recording profile contains all relevant information to enable a recording session for a specific analysis scenario. This includes configuration information about ETW sessions and providers. Each `.samp` file contains a recording profile definition, which among other things, associates a specific set of ETW providers with a recording session.

SAM by default delegates records provided by a configured ETW provider (i.e., configured in a recording profile) into Windows Event Log[4] system through an ETW provider with the name `SystemActivityMonitor`. The records recorded by SAM can be viewed with the Windows Event Viewer utility at the Event Viewer path `Application and Services Logs/SystemActivityMonitor/Operational`.

An ETW record always consists of a record header and additional record data (see section 2.3). Figure 2 shows the record header and additional record data written by SAM, displayed by the Windows Event Viewer in XML format.



```
<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
  <System>
      <Provider Name="SystemActivityMonitor" Guid="{4eb5f182-ae41-4822-929a-f7b91a7a3eab}" />
      <EventID>10</EventID>
      <Version>1</Version>
      <Level>4</Level>
      <Task>0</Task>
      <Opcode>10</Opcode>
      <Keywords>0x8000000000000001</Keywords>
      <TimeCreated SystemTime="2021-11-16T13:33:50.5466329Z" />
      <EventRecordID>253869361</EventRecordID>
      <Correlation />
      <Execution ProcessID="2824" ThreadID="22184" />
      <Channel>SystemActivityMonitor/Operational</Channel>
      <Computer>DESKTOP-CMHQP24</Computer>
      <Security UserID="S-1-5-21-3997305478-3333273705-1638080041-1001" />
  </System>
  <EventData>
      <Data Name="ProviderName">Microsoft-Windows-Kernel-Network</Data>
      <Data Name="ProviderGUID">{7dd42a49-5329-4832-8dfd-43d979153a88}</Data>
      <Data Name="EventID">11</Data>
      <Data Name="EventName" />
      <Data Name="EventDecodingSource">DecodingSourceXMLFile</Data>
      <Data Name="EventVersion">0</Data>
      <Data Name="EventLevel">4</Data>
      <Data Name="EventTask">10</Data>
      <Data Name="EventOpcode">11</Data>
      <Data Name="EventOpcodeName">Data received.</Data>
      <Data Name="EventKeyword">0x8000000000000010</Data>
      <Data Name="EventTime">132814730267022830</Data>
      <Data Name="EventUTCTime">2021-11-15T18:03:46.7022830Z</Data>
      <Data Name="ImageName">pwsh.exe</Data>
      <Data Name="CommandLine">C:\Users\dphillips\AppData\Local\Microsoft\WindowsApps\Microsoft.PowerShellPreview_8wekyb3d8bbwe\pwsh.exe</Data>
      <Data Name="ProcessId">972</Data>
      <Data Name="ThreadId">21052</Data>
      <Data Name="ParentImageName">WindowsTerminal.exe</Data>
      <Data Name="ParentCommandLine">"C:\Program Files\WindowsApps\Microsoft.WindowsTerminalPreview_1.12.2931.
      0_x64__8wekyb3d8bbwe\WindowsTerminal.exe"</Data>
      <Data Name="ParentProcessId">11920</Data>
      <Data Name="EventPayload">{"PID":"5720","size":"608","daddr":"142.250.186.78","saddr":"10.0.2.15","dport":"443","sport":"63257",
      "seqnum":"0","connid":"0"}</Data>
  </EventData>
</Event>
```

1
record header
3
encapsulated record header
2
additional record data
4
encapsulated additional record data

Figure 2: Example record written by SAM

[4] https://docs.microsoft.com/en-us/windows/win32/wes/windows-event-log [Retrieved:12.08.2021]

The record header contains meta information, such as record ID, name of SAM's internal ETW provider, the time the record was created by SAM, the process ID of the SAM process, and so on (see, Figure 2, label 1). The additional record data contains the encapsulated data provided by a configured ETW provider i.e., the header and additional record data of this provider (see Figure 2, label 2). The data fields of the encapsulated record header are presented as individual attributes (see Figure 2, label 3), whereas data fields of the encapsulated additional record data are serialized into JavaScript Object Notation (JSON) presented as single attribute (see Figure 2, label 4). Serializing the additional record data of a configured ETW provider as single JSON formatted attribute enables SAM to provide a single data format and layout, that is standardized and independent of the underlying ETW provider providing record data.

It must be emphasized that strings written to the Windows Event Log are limited to a size of 32K characters. For this purpose, SAM distinguishes between recorded records that exceed this limit or not. Because a record which exceed this limit have to be divided into several contiguous records, otherwise it cannot be displayed by the Windows Event Log. Records that are below this limit are witten to the Windows Event Log with `EventID 10`, whereas records that exceed the limit are written with `EventID 20`. The formate and layout are conceptional identical except for the `Data Name="MessageNumber"` and `Data Name="MessageTotal"` fields, which are nessesary to determine the order of a contiguous record set. Table 1 summarizes and describes each of the fields available in a record that SAM writes to the Windows event log.

	Field	**Description**
System	Provider Name="[Value]"	Defines the name of SAM's internal ETW provider.
	Provider Guid="[Value]"	Defines the GUID of SAM's internal ETW provider.
	EventID	Defines the unique number of a specific event type written by SAM.
	Version	Defines the event version that the SAM provider attaches to events.
	Level	Defines the event level that can be used for filtering that the SAM provider attaches to events
	Task	Defines the tasks that apply to the event that the SAM provider attaches to events
	Opcode	Defines the standard operation codes that the SAM provider attaches to events.
	Keywords	Defines the event keyword that can be used for filtering that the SAM provider attaches to events.
	TimeCreated SystemTime="[Value]"	Defines the timestamp at which the record was created by SAM in UTC timestamp format.
	EventRecordID	Defines the number for the record written by SAM. The very first record written to the Windows Event Log is record number 1, and other records are numbered sequentially.
	Execution ProcessID="[Value]"	Defines the process ID of the SAM process from which the record is derived.

	Field	Description
System	Execution ThreadID="[Value]"	Defines the thread ID running in the context of the SAM process from which the record is derived.
	Channel	Defines the event channel name in which SAM writes recorded records.
	Computer	Defines the computer name from which SAM recorded the record.
	Security UserID="[Value]"	Defines the security identifier of the SAM process from which the record is derived.
EventData	Data Name="MessageNumber" (appears only in complex records (i.e. `EventID 20`))	Defines the index number of a single record of a record set (i.e. excessively large record that had to be divided).
	Data Name="MessageTotal" (appears only in complex records (i.e. `EventID 20`))	Defines the total number of a record set.
	Data Name="ProviderName"	Defines the name of the ETW provider.
	Data Name="ProviderGUID"	Defines the GUID for the ETW provider.
	Data Name="EventID"	Defines the unique number for a specific event type.
	Data Name="EventName"	Defines the unique name for specific event Type.
	Data Name="DecodingSource"	Defines the decoding source of the event data.
	Data Name="EventVersion"	Defines the event version.
	Data Name="EventLevel"	Defines the event level that can be used for filtering.
	Data Name="EventTask":	Defines the tasks that apply to events.
	Data Name="EventOpcode"	Defines the standard operation codes that the event source attaches to events.
	Data Name="EventOpcodeName"	Defines the operation code name.
	Data Name=" EventKeyword"	Defines the event keyword that can be used for filtering.
	Data Name="EventTime"	Defines the timestamp at which the record was created as unsigned 64-bit integer.
	Data Name="EventSystemTime"	Defines the timestamp at which the record was created in UTC timestamp format.
	Data Name="ImageName"	Defines the process image name from which the record is derived.
	Data Name="CommandLine"	Defines the command line parameters of the process from which the record is derived.

	Field	Description
EventData	Data Name="ProcessId"	Defines the process ID from which the record is derived.
	Data Name="ThreadId"	Defines the thread ID from which the record is derived.
	Data Name="ParentImageName"	Defines the parent process image name of the process from which the record is derived.
	Data Name="ParentCommandLine"	Defines the command line parameters of the parent process from which the record is derived.
	Data Name="ParentProcessId"	Defines the parent process ID of the process from which the record is derived.
	Data Name="EventPayload"	Defines the encapsulated additional record data derived from a ETW provider.

Table 1: Field description of a record written by SAM into the Windows Event Log

3.1 Installation and Usage

This section discusses how SAM is installed on a system and which system modifications are conducted during the installation of SAM. In addition, it also discusses the two primary usage scenarios that enable an analyst to view and further process the recorded data.

Installing SAM[5]: SAM can be installed on a Windows system using a `.msi` package (`msiexec.exe /i setup.msi /qn`)[6]. This allows SAM to be installed on the system, which includes:

- Copying necessary files on the filesystem: By default, SAM files are located at `C:\Program Files\SystemActivityMonitor\Sam` path.
- Creation of an Event channel: `Application and Services Logs/SystemActivityMonitor`
- Registration of a SAM ETW provider: `SystemActivityMonitor(4eb5f182-ae41-4822-929a-f7b91a7a3eab)`

In order to install SAM on a system the following system requirement has to be met:

- Supported operating system and C++ redistributable package: Windows 10 version 1607 or higher and Visual C++ Redistributable for Visual Studio 2019

Local Usage Scenario: After a successful installation, SAM is ready for use. Once started SAM writes its records into the Windows Event Log channel `Application and Services Logs/SystemActivityMonitor/Operational`. The records stored in the channel can then be processed as usual e.g., viewed and analyzed locally with the Windows Event Viewer or PowerShell utility. As already mentioned, SAM serializes the additional record data provided by a configured ETW provider into a single JSON formatted attribute. However, the Windows Event Viewer utility does not have JSON parsing capabilities, which makes this utility difficult for a more in-depth analysis (e.g., filtering records based on parameter stored as part of the JSON string). Therefore, it is recommended to use the PowerShell

[5] The installation must be performed with local administrator rights.
[6] If there are any issues during installation try a verbose installation method: `start /wait msiexec.exe /i setup.msi /qn /l sam.log`

utility for a more in-depth local analysis, as it provides significantly better and more flexible data processing capabilities.

Example #PowerShell demonstrates the analysis of records stored in the Windows Event Log channel `Application and Services Logs/SystemActivityMonitor/Operational`. The analysis is conducted using the `Get-WinEvent` and `ConvertFrom-Json` PowerShell commands, in the context of a PowerShell script, to extracted records from the Windows Event Log channel `Application and Services Logs/SystemActivityMonitor/Operational` associated with a specific ETW provider and event ID. In this context, the `Get-WinEvent` command is used to extract the records from the Windows Event Log channel, whereas the `ConvertFrom-Json` is used to deserialize the JSON string. In the end all data (i.e., header and additional record data) is copied into a custom PowerShell object, which now makes them easily accessible for further processing.

Example #PowerShell

```
function Get-SAMEventsPayload()
{
  [CmdletBinding()]
  Param
  (
    [Parameter(ValuefromPipeline=$true, Position = 1, Mandatory = $false)]
    [String]$ProviderName,
    [Parameter(ValuefromPipeline=$true, Position = 2, Mandatory = $false)]
    [String]$EventName,
    [Parameter(ValuefromPipeline=$true, Position = 3, Mandatory = $false)]
    [Int64]$EventId,
    [Parameter(ValuefromPipeline=$true, Position = 4, Mandatory = $false)]
    [Int64]$MaxEvents = 1000
  )

  $EventsAsXmlObject = Get-WinEvent -LogName "SystemActivityMonitor/Operational" -MaxEvents
$MaxEvents | ForEach-Object{ ([xml]$_.ToXml())}

  if($ProviderName)
  {
    $_EventsAsXmlObject = $EventsAsXmlObject.Where({
      ($_.Event.EventData.Data.Name -eq "ProviderName") -and ($_.Event.EventData.Data.InnerText -
eq $ProviderName)
    })

    $EventsAsXmlObject = $_EventsAsXmlObject
  }

  if($EventName)
  {
    $_EventsAsXmlObject = $EventsAsXmlObject.Where({
      ($_.Event.EventData.Data.Name -eq "EventName") -and ($_.Event.EventData.Data.InnerText -eq
$EventName)
    })

    $EventsAsXmlObject = $_EventsAsXmlObject
  }

  if($EventId)
  {
    $_EventsAsXmlObject = $EventsAsXmlObject.Where({
      ($_.Event.EventData.Data.Name -eq "EventID") -and ($_.Event.EventData.Data.InnerText -eq
$EventId)
    })
    $EventsAsXmlObject = $_EventsAsXmlObject
  }

  foreach($EventAsXmlObject in $EventsAsXmlObject)
  {
    $EventObjectAsPsObject = New-Object -TypeName PSObject
    foreach($EventDataAsXmlElement in $EventAsXmlObject.Event.EventData.Data)
    {
      if($EventDataAsXmlElement.Name -eq "EventPayload")
      {
        foreach($EventPayloadProperty in ($EventDataAsXmlElement.InnerText | ConvertFrom-
Json).PSObject.Properties)
        {
          $PropertyName = $EventDataAsXmlElement.Name + "-" + $EventPayloadProperty.Name
          Add-Member -InputObject $EventObjectAsPsObject -MemberType NoteProperty -Name
$PropertyName -Value $EventPayloadProperty.Value
        }
      }
      else
      {
        Add-Member -InputObject $EventObjectAsPsObject -MemberType NoteProperty -Name
$EventDataAsXmlElement.Name -Value $EventDataAsXmlElement.InnerText
      }
    }

    $EventObjectAsPsObject
  }
}
Get-SAMEventsPayload -ProviderName "Microsoft-Windows-DNS-Client" -EventId 3008 -MaxEvents 10
```

Output

```
=====> PowerShell Integrated Console v2021.12.0 <=====
PS C:\Users\dphillips> c:\Users\dphillips\Documents\Get-SAMEventsPayload.ps1
ProviderName              : Microsoft-Windows-DNS-Client
ProviderGUID              : {1c95126e-7eea-49a9-a3fe-a378b03ddb4d}
EventID                   : 3008
EventName                 :
EventDecodingSource       : DecodingSourceXMLFile
EventVersion              : 0
EventLevel                : 4
EventTask                 : 0
EventOpcode               : 0
EventOpcodeName           :
EventKeyword              : 0x8000000000000000
EventTime                 : 132842325190763194
EventUTCTime              : 2021-12-17T16:35:19.0763194Z
ImageName                 : svchost.exe
CommandLine               : C:\Windows\System32\svchost.exe -k utcsvc -p
ProcessId                 : 7140
ThreadId                  : 16536
ParentImageName           : services.exe
ParentCommandLine         :
ParentProcessId           : 716
EventPayload-QueryName    : v10.events.data.microsoft.com
EventPayload-QueryType    : 28
EventPayload-QueryOptions : 2251800887582720
EventPayload-QueryStatus  : 1460
```

```
EventPayload-QueryResults :
...
```

Code Block 1: Query SAM generated records via PowerShell

Remote Usage Scenario: An alternative to the local analysis approach described above is the use of a comprehensive log management and analysis infrastructure to analyze the records, such as Elastic Stack. As already mentioned, once started SAM delegates records into the Windows Event Log channel `Application and Services Logs/SystemActivityMonitor/Operational`. Forwarding these records to a log management and analysis infrastructure offers several advantages over a local data analysis with e.g., the PowerShell utility. Such an infrastructure is specifically designed to process and manage large amounts of record data. Therefore, they are equipped by default with many built-in features, such as the quick search or aggregation of stored record data in order to identify trends and patterns. However, the specific procedure and processing options depend on the environment, implementation, and the used log management and analysis infrastructure.

3.2 SAM How To Topics

This section provides procedures on how to use SAM. A recording session can be managed or unmanaged. Managed means registering and executing SAM as Windows service, whereas unmanaged means executing SAM as a high privileged user from a `cmd.exe` or `powershell.exe` session. Therefore, the following procedures will now be discussed in more detail: start a recording, stop a recording, managing a recording profile, and authoring of a recording profile.

Start a Recording: This procedure describes how to start a recording session. Starting a managed recording session requires that SAM is executed as Windows service. In order to start SAM as Windows service, SAM needs to be registered as Windows service. The registering and unregistering of SAM as Windows service is conducted with the following commands: `sam.exe -register_service <recording profile id>` and `sam.exe -ungregister_service`.

Once SAM was registered as Windows service it can be managed (i.e., started or stopped) using Windows build in procedures. For example, by using the PowerShell cmdlets `start-service -name sam` and `stop-service -name sam`.

As already mentioned, SAM can also be started unmanaged. This requires that SAM is executed from a high privileged (i.e., administrator privileges) `cmd.exe` or `powershell.exe` session. Starting an unmanaged recording session is conducted with the following command: `sam.exe -start <recording profile id>`.

Stop a Recording: This procedure describes how to stop a managed or unmanaged recording session[7]. Stopping an unmanaged recording session is conducted with shortcut `CTRL+Q`, whereas a managed recording session is stopped by stopping the registered SAM service e.g., with the PowerShell command `stop-service -name sam`.

Managing a Recording Profile: This procedure describes how to add, remove, and list SAM recording profiles. As already discussed, SAM recording profiles control all aspects of a recording session. This includes configuration information about ETW sessions and providers.

- Add a recording profile: `sam.exe -add <recording profile file (fullpath)>`
- Remove a recording profile: `sam.exe -remove < recording profile id>`
- List stored recording profiles and the associated profile ID: `sam.exe -list`

[7] It is important to emphasize that it can sometimes take a short time to end a recording session because of the need to process data that is already in the processing pipeline.

- Update the recording profile of a running managed recording session: `sam.exe –update_config <recording profile id>`

3.3 Authoring Recording Profiles

This section describes all XML elements that are used to author a SAM recording profile. Recording profiles contain all the necessary information to conduct a recording session for a specific analysis scenario. This includes information about ETW sessions, providers, and keywords. An example of such a file and the XML schema definition of a recording profile is placed in the Appendix, section 'Example #Recording Profile' and 'SAMRecordingProfile.xsd'.

A SAM recording profile supports the following ETW features:

- User- and kernel-land ETW sessions
- User-land ETW provider and attributes, such as name and GUID, detail level, and keywords
- Kernel-land ETW flags
- Filter rules to control the recording of records.

The following represents the element hierarchy of a SAM recording profile.

```
<SystemActivityMonitor>
  <Collector>
    <SystemCollector>[...]</SystemCollector>
    <EventCollector>[...]</EventCollector>
    <Filter>[...]</Filter>
  </Collector>
  <Provider>
    <SystemProviders>[...]</SystemProviders>
    <EventProviders>[...]</EventProviders>
  </Provider>
</SystemActivityMonitor>
```

Code Block 2: General SAM recording profile XML node overview

Collector: SAM supports two types of collectors: the *SystemCollector* and *EventCollector*. The *SystemCollector* definition specifies buffer sizes and other attributes for ETW kernel-land recording sessions, whereas the *EventCollector* definition specifies buffer sizes and other attributes for ETW user-land recording sessions. A collector definition requires a mandatory `Name` attribute. The `Name` attribute specifies the unique ETW session name (e.g., `SamSystemCollector`) that should be enabled. In addition, the following optional attributes can be used to fine-tune *SystemCollector* and *EventCollector* parameters:

- `FlushTime`: Specifies the time interval at which any non-empty buffers are flushed.
- `BuffersSize`: Specifies the size of each single allocated buffer, in kilobytes.
- `MinimimBuffers`: Specifies the minimum number of buffers to be allocated when starting a session.
- `MaximumBuffers`: Specifies the maximum number of buffers to be allocated when starting a session.

Example #Collector demonstrates a *SystemCollector* and *EventCollector* definition.

<u>Example #Collector</u>

```
<SystemCollector Name="SamSystemCollector">
  <FlushTimer Value="1"/>
  <BufferSize Value="1024"/>
  <MinimumBuffers Value="80"/>
  <MaximumBuffers Value="128"/>
</SystemCollector>
```

```xml
<EventCollector Name="SamEventCollector">
  <FlushTimer Value="1"/>
  <BufferSize Value="1024"/>
  <MinimumBuffers Value="80"/>
  <MaximumBuffers Value="128"/>
</EventCollector>
```

Code Block 3: XML collector node example

A global filter can be defined at the collector level to specify a list of command line or image name field rules that can be used to filter out records that match a rule. A filter as well as the field rules contain conditions that correspond to a value. This includes: `condition="include"` the filter defines whether records are included that match a rule, `condition="exclude"` the filter defines whether records are excluded that match a rule, `condition="is"` the field equals the value, and `condition="contains"` the field contains the value.

Example #Filter demonstrates a collector *Filter* definition.

Example #Filter

```xml
<Filter condition="exclude">
  <CommandLine condition="is">svchost.exe</CommandLine>
  <CommandLine condition="contains">svchost</CommandLine>
  <ImageName condition="is">svchost.exe</ImageName>
  <ImageName condition="contains">diagtrack</ImageName>
</Filter>
```

Code Block 4: XML collector filter node example

Provider: SAM supports two types of providers: the *SystemProvider* and *EventProvider*. The *SystemProvider* definition specifies the system keywords to use in an ETW kernel-land recording sessions. The *EventProvider* definition specifies the provider to use and the keywords and levels to enable in an ETW user-land recording sessions.

A *SystemProvider* definition requires a mandatory `Name` attribute. The `Name` attribute describes the kernel-land ETW flag that should be enabled for the kernel-land session. In addition, the following optional attributes can be used to fine-tune *SystemProvider* parameters:

- `Filter:` Specifies a list of either event ID or opcode values that can be used to filter out records that match a filter. A filter contains conditions that correspond to a value. This includes: `condition="include"` the filter defines whether records are included that match a value and `condition="exclude"` the filter defines whether records are excluded that match a value.

An *EventProvider* definition requires a mandatory `Name` and `Guid` attribute. The `Name` attribute describes the registered unique ETW provider name (e.g., `Microsoft-Windows-Kernel-Process`), whereas the `Guid` attribute describes the registered unique ETW provider GUID (e.g., `22FB2CD6-0E7B-422B-A0C7-2FAD1FD0E716`) that should be enabled for the user-land session. In addition, the following optional attributes can be used to fine-tune *EventProvider* parameters:

- `Level`: Specifies a 1-byte ETW logging level (i.e. value range from `0` to `255`) value associated with a record. This provides the ability to filter records based on the associated logging level.
- `Any` and `ALL`: Specifies an 8-byte hexadecimal bitmask (i.e. value range from `0x0000000000000000` to `0xffffffffffffffff`) value that can be used to filter out records that correspond to the bits set in the bitmask.

  For example, ETW provider `X` supports the three keywords (i.e. `READ: 0x001`; `WRITE: 0x010`; and `EXECUTE: 0x100`) and two records (i.e. record 1 -> "`READ` + `WRITE`" keywords (`0x011`); and record 2 -> "`READ` + `EXECUTE`" keywords (`0x101`)).

To filter out any record that has an associated `READ` keyword, the `Any` attribute (i.e. match ***any*** of the bit set in the bitmask) can be set to `0x001`. In this case, the ETW provider `X` will produce and deliver record 1 and 2. In order to filter out only records that have an associated `READ` + `EXECUTE` keyword, the `All` attribute (i.e. match ***all*** of the bit set in the bitmask) can be set to `0x101`. In this case, the ETW provider `X` will produce and deliver only record 2.

- `Rundown`: Specifies a Boolean value that controls if rundown records should be captured. Rundown records are not true real-time tracing records. Instead, they describe the state of the system - either at the start or at the end of a trace session.
- `Filter`: Specifies a list of either event ID or opcode values that can be used to filter out records that match a filter. A filter contains conditions that correspond to a value. This includes: `condition="include"` the filter defines whether records are included that match a value and `condition="exclude"` the filter defines whether records are excluded that match a value.

Example #Provider demonstrates a system and event provider definition.

Example #Provider

```
<SystemProviders>
  <SystemProvider Name="DiskIO">
    <Filter condition="include">
      <EventIdList>1 5 64</EventIdList>
                 <!-- either use EventIdList or OpcodeIdList -->
      <OpcodeIdList>7 2 9</OpcodeIdList>
    </Filter>
  </SystemProvider>
</SystemProvider>

<EventProviders>
  <EventProvider
    Name="Microsoft-Windows-Kernel-EventTracing"
    Guid="{b675ec37-bdb6-4648-bc92-f3fdc74d3ca2}"
    Level="255"
    Any="0x010"
    All="0x020"
    Rundown="true">
    <Filter condition="exclude">
      <EventIdList>1 5 64</EventIdList>
                 <!-- either use EventIdList or OpcodeIdList -->
      <OpcodeIdList>7 2 9</OpcodeIdList>
    </Filter>
  </EventProvider>
</EventProviders>
```

Code Block 5: XML provider node example

3.4 Avoiding Lost Events

As already discussed, SAM uses the extremely low overhead and high speed ETW infrastructure to collect behavior and resource usage data. However, depending on the underlaying hardware and the deployed recording profile there are certain situations where an ETW provider or a collection of ETW providers generate so many records that the ETW infrastructure cannot keep up with the recording frequency. This problem manifests as lost records in a recording session and could lead to analysis difficulties or erroneous conclusions because of incomplete datasets. In order to avoid situations in which record data is lost, the following basic rules should be considered when creating a recording session:

- Simplify the analysis scenario e.g., select a fewer number of ETW providers.
- Increase the number of buffers.
- Increase the size of buffers

- Use advanced hardware e.g., use a system with a powerful CPU for higher throughput. This is the last option to consider. A better option to avoid losing records is carefully selecting the providers to enable and the buffers to use.

4 Windows Telemetry Component Profile

This section discusses an exemplary deployment of a Windows Telemetry component profile i.e., the detailed recording and analysis of the behavior and the resource usage of the Windows Telemetry component (also referred to as DiagTrack in this work) in a specific scenario with the help of SAM. The recording is based on the recording profile (see Appendix, section 'Exemplary DiagTrack Recording Profile') which controls all aspects of the recording session. However, the recorded data is analyzed using the Elastic Stack log management and analysis infrastructure. Therefore, an environment consisting of three Windows systems was prepared. Figure 3 depicts this environment which includes: one Windows 10 system and two Windows server systems, and several additional configured or installed software components.

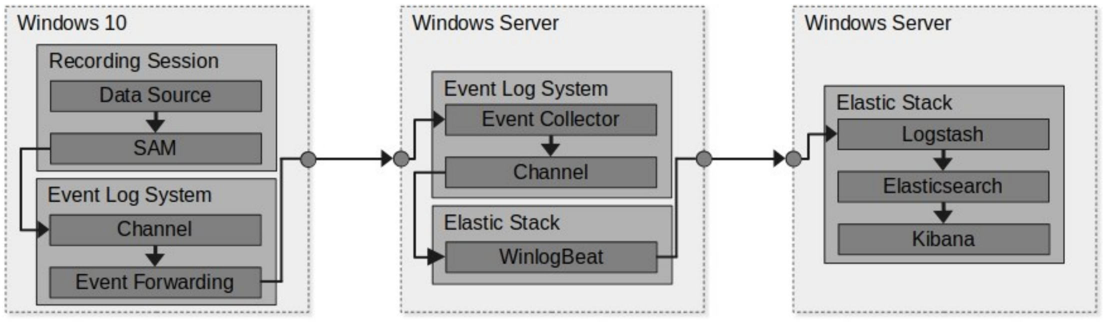


Figure 3: Exemplary Elastic Stack analysis environment

Table 2 lists each of these systems, its purpose, and the additional configured or installed software.

System Purpose	Description	Additional Configured and Installed Software
Windows 10 system hosting the DiagTrack service and the SAM recording session.	Windows 10 system on which relevant system activities are recorded with SAM in order to analyze the behavior and resource usage of the DiagTrack component. SAM delegates records into the Windows Event Log channel `Application and Services Logs/SystemActivityMonitor/Operational`, from where they are further processed using Event Forwarding (i.e., send to the Event Collector).	• Windows Event Forwarding • SAM (incl. corresponding recording profile)
Windows Server system hosting the Windows Event Collector and WinlogBeat components.	The Windows Server system receives the recorded activities (i.e., acting as Event Collector) from the Windows 10. Once arrived, they are further processed with the help of the WinlogBeat[8] component (i.e., shipped to the Elasticsearch or Logstash).	• Windows Event Collector • WinlogBeat
Windows Server system hosting the Elastic Stack infrastructure	Windows Server system which operates the Elastic Stack log management and analysis infrastructure, to which the Windows 10 recorded activities are ultimately forwarded. Once arrived, the built-in features of the Elastic Stack components can be used to perform the actual component profile analysis.	• Logstash • Elasticsearch • Kibana

Table 2: Overview of configured systems

[8] https://www.elastic.co/beats/winlogbeat [Retrieved:12.08.2021]

The selection of ETW providers defined in the SAM recording profile (see Appendix, section ‘Exemplary DiagTrack Recording Profile’) is based on activities between the relevant aspects of operating system and DiagTrack operation: interaction between DiagTrack and the operating system components, initialization of DiagTrack and operation of DiagTrack. Table 1 provides an overview of the scenarios considered in this component profile i.e., an overview of scenario descriptions (column 'Description') and ETW provider associated with the scenario (column ETW Provider').

Description	ETW Provider

Activities related to the operation of processes, such as process start and stop and image loading.	Name=`Microsoft-Windows-Kernel-Process`
	Guid=`22FB2CD6-0E7B-422B-A0C7-2FAD1FD0E716`

Activities related to the operation of the ETW infrastructure, such as ETW provider registration and ETW session start.	Name=`Microsoft-Windows-Kernel-EventTracing`
	Guid=`b675ec37-bdb6-4648-bc92-f3fdc74d3ca2`

Activities related to network operation, such as connection establishment and termination.	Name=`Microsoft-Windows-Kernel-Network`
	Guid=`7dd42a49-5329-4832-8dfd-43d979153a88`

Activities related to the operation of registry, such as registry read and write.	Name=`Microsoft-Windows-Kernel-Registry`
	Guid=`70eb4f03-c1de-4f73-a051-33d13d5413bd`

Activities related to the operation of the diagtrack component, such as collected telemetry data.	Name= `Microsoft-Windows-Diagtrack`
	Guid= `43ac453b-97cd-4b51-4376-db7c9bb963ac`

Activities related to DNS operation, such as IP to Domain name resolution.	Name=`Microsoft-Windows-DNS-Client`
	Guid=`1c95126e-7eea-49a9-a3fe-a378b03ddb4d`

Activities related to the operation of files, such as file read and write.	Name=`Microsoft-Windows-Kernel-File`
	Guid=`edd08927-9cc4-4e65-b970-c2560fb5c289`

Table 3: Overview of considered component profile scenarios

4.1 Exemplary Data Analysis

This section provides a brief overview of two exemplary analysis scenarios in tabular form. A scenario (indicated by the symbol ⊜) is either a single record or a correlation of multiple records. A scenario that is based on the correlation of multiple records, combines multiple records into an aggregation of records and thus of activities. When a scenario consists of two or more correlated records, the symbol ⊕ indicates the record correlation. The symbol appears between the ETW provider, that produces the records to be correlated. If records need to be filtered, the symbol ✂ indicate a record filter – for example, to reduce high volume records.

Uploaded Telemetry Data (single record scenario): Diagtrack collects telemetry data in a JSON-structured form. Once data is sent to the Microsoft’s backend infrastructure, the ETW provider `Microsoft.Windows.DiagTrack`
generates the record `AsimovUploader_PersistEvent` containing the sent telemetry data.

Scenario	Track uploaded telemetry data.
⊜	
ETW Provider	`Microsoft.Windows.DiagTrack`

✂	`winlog.event_data.ProviderName:"Microsoft.Windows.DiagTrack" AND winlog.event_data.EventName:"AsimovUploader_PersistEvent"`
Microsoft.Windows.DiagTrack EventID: 0 (AsimovUploader_PersistEvent)	`{"EventPayload": "{"ver":"4.0","name":"Microsoft.OSG.DU.DeliveryOptClient.TraceRoute","time":"2021-08-05T10:54:58.3583876Z","iKey":"o:0a89d516ae714e01ae89c96d185e9ae3","ext":{"utc":{"eventFlags":258,"pgName":"WIN","flags":472908336,"epoch":"1009422","seq":6878},"metadata":{"f":{"startTime":5}},"os":{"bootId":9,"name":"Windows","ver":"10.0.19042.1110.amd64fre.vb_release.191206-1406"},"app":{"id":"W:0000f519feec486de87ed73cb92d3cac802400000000!`

	0000010db07461e45b41c886192df6fd425ba8d42d82!svchost.exe","ver":"1972/12/14:16:22:50!1C364!svchost.exe","asId":11857},"device":{"localId":"s:B94EF8EB-7A52-4FAF-82CC-F4C5C23806B9","deviceClass":"Windows.Desktop"},"protocol":{"devMake":"LENOVO","devModel":"10MKCTO1WW"},"user":{"localId":"w:E3111673-C112-BB48-A6E4-98589113B47D"},"loc":{"tz":"+02:00"}},"data":{"experimentId":0,"startTime":132726342382031145,"networkType":"Ethernet","destIP":"205.185.216.10","hopInfoList":[{"IP":"109.90.147.0","RTT":1,"hopNumber":3},{"IP":"81.210.148.0","RTT":12,"hopNumber":5},{"IP":"84.116.191.221","RTT":12,"hopNumber":6},{"IP":"84.116.190.94","RTT":10,"hopNumber":7},{"IP":"151.139.84.6","RTT":10,"hopNumber":8},{"IP":"151.139.84.14","RTT":10,"hopNumber":9},{"IP":"94.46.154.139","RTT":17,"hopNumber":10}],"deviceProfile":1310976}}","EventLatency": "0","EventPersistence": "1","GroupId": "{4f50731a-89cf-4782-b3e0-dce8c90476ba}","ProviderId": "{f13cd440-a9b2-4d51-8b09-62dc6ad60194}","Categories": "140737488355328","IsCore": "0","ProcessId": "3020","StorageBufferType": "0","EventSizeBytes": "1247"}

Table 4: Track uploaded telemetry data

Downloaded Settings Data (multiple records scenario): DiagTrack downloads settings data and stores the data in the filesystem (e.g., the file %ProgramData%\Microsoft\Diagnosis\DownloadedSettings\utc.app.json, see section (ERNW_WP4)). The ETW provider Microsoft.Windows.DiagTrack generates the record AttemptingDownload each time a download attempt is initiated and a SettingsDownloader_FinishDownload once the download is finished. Correlating these records with those of the Microsoft-Windows-Kernel-File ETW provider, allows to track when settings data is downloaded and where this date is stored in the filesystem.

Scenario	Track downloaded settings data.
⊜	
ETW Provider	Microsoft.Windows.DiagTrack and
⊕	Correlating Microsoft.Windows.DiagTrack (i.e., AttemptingDownload and SettingsDownloader_FinishDownload) with Microsoft-Windows-Kernel-File activities.
ETW Provider	Microsoft-Windows-Kernel-File

✂	((winlog.event_data.ProviderName:"Microsoft-Windows-Kernel-File" OR winlog.event_data.EventName:AttemptingDownload) AND winlog.event_data.EventPayload:utc.app*) OR winlog.event_data.EventName:"SettingsDownloader_FinishDownload"
Microsoft.Windows.DiagTrack EventID: 0 (AttemptingDownload)	{"Endpoint": "utc.app"}
Microsoft-Windows-Kernel-File EventID: 12 (FileObjectCreate)	{"Irp": "0xFFFFE489489F20F8","FileObject": "0xFFFFE4893BF8D650","IssuingThreadId": "12032","CreateOptions": "0x1200000","CreateAttributes": "0x0","ShareAccess": "0x7","FileName": "\Device\HarddiskVolume3\ProgramData\Microsoft\Diagnosis\DownloadedSettings\utc.app.json"}
Microsoft-Windows-Kernel-File EventID: 12 (FileObjectCreate)	{"Irp": "0xFFFFE489489F20F8","FileObject": "0xFFFFE4893BF8B3F0","IssuingThreadId": "12032","CreateOptions": "0x5000060","CreateAttributes": "0x100","ShareAccess": "0x1","FileName": "\Device\HarddiskVolume3\ProgramData\Microsoft\Diagnosis\DownloadedSettings\utc.app.json.new"}
Microsoft-Windows-Kernel-File EventID: 30 (CreateNewFile)	{"Irp": "0xFFFFE489489F20F8","FileObject": "0xFFFFE4893BF8B3F0","IssuingThreadId": "12032","CreateOptions": "0x5000060","CreateAttributes": "0x100","ShareAccess": "0x1","FileName": "\Device\HarddiskVolume3\ProgramData\Microsoft\Diagnosis\DownloadedSettings\utc.app.json.new"}
Microsoft-Windows-Kernel-File EventID: 10 (NameCreate)	{"FileKey": "0xFFFFBD864CC1E7C0","FileName": "\Device\HarddiskVolume3\ProgramData\Microsoft\Diagnosis\DownloadedSettings\utc.app.json.new"}
Microsoft-Windows-Kernel-File EventID: 22 (QueryInformation)	{"Irp": "0xFFFFE48946BED0F8","FileObject": "0xFFFFE4893BF8B3F0","FileKey":

	"0xFFFFBD864CC1E7C0","ExtraInformation": "0x0","IssuingThreadId": "12032","InfoClass": "5"}
Microsoft.Windows.DiagTrack EventID: 0 (SettingsDownloader_FinishDownload)	{"Partner": "","Feature": "","Headers": "","ErrorCode": "3997765","DownloadDurationMs": "3659436697583654","OnDiskFileSizeBytes": "10"}
Microsoft-Windows-Kernel-File EventID: 18 (SetDelete)	{"Irp": "0xFFFFE48946BED0F8","FileObject": "0xFFFFE4893BF8B3F0","FileKey": "0xFFFFBD864CC1E7C0","ExtraInformation": "0x1","IssuingThreadId": "12032","InfoClass": "13"}
Microsoft-Windows-Kernel-File EventID: 14 (FileObjectClose)	{"Irp": "0xFFFFE48946BED0F8","FileObject": "0xFFFFE4893BF8B3F0","FileKey": "0xFFFFBD864CC1E7C0","IssuingThreadId": "12032"}
Microsoft-Windows-Kernel-File EventID: 11 (NameDelete)	{"FileKey": "0xFFFFBD864CC1E7C0","FileName": "\Device\HarddiskVolume3\ProgramData\Microsoft\Diagnosis\DownloadedSettings\utc.app.json.new"}
Microsoft-Windows-Kernel-File EventID: 22 (QueryInformation)	{"Irp": "0xFFFFE48946BED0F8","FileObject": "0xFFFFE4893BF8B3F0","FileKey": "0xFFFFBD86519097C0","ExtraInformation": "0x0","IssuingThreadId": "12032","InfoClass": "4"}

Table 5: Track downloaded settings data.

Appendix

Example #Recording Profile

Example #Recording Profile

```xml
<?xml version='1.0' encoding='utf-8' standalone='yes'?>
<SystemActivityMonitor Author="Dominik Phillips" Company="ERNW" Team="Windows Security"
Version="1.0">
  <Collector>
    <SystemCollector Name="SAM_KERNEL_SESSION">
      <BufferSize Value="1024"/>
      <MinimumBuffers Value="64"/>
      <MaximumBuffers Value="128"/>
      <FlushTimer Value="1"/>
    </SystemCollector>
    <EventCollector Name="SAM_USER_SESSION">
      <BufferSize Value="1024"/>
      <MinimumBuffers Value="64"/>
      <MaximumBuffers Value="128"/>
      <FlushTimer Value="1"/>
    </EventCollector>
  </Collector>
  <Provider>
    <SystemProviders>
      <SystemProvider Name="DiskIO">
      </SystemProvider>
      <SystemProvider Name="FileIO">
      </SystemProvider>
      <SystemProvider Name="FileIOInit">
      </SystemProvider>
      <SystemProvider Name="HardFaults">
      </SystemProvider>
    </SystemProviders>
    <EventProviders>
      <EventProvider Name="Microsoft-Windows-Kernel-EventTracing" Guid="{b675ec37-bdb6-4648-bc92-
f3fdc74d3ca2}" Rundown="true">
      </EventProvider>
      <EventProvider Name="Microsoft-Windows-Kernel-Network" Guid="{7dd42a49-5329-4832-8dfd-
43d979153a88}" Rundown="true">
      </EventProvider>
      <EventProvider Name="Microsoft-Windows-Kernel-Registry" Guid="{70eb4f03-c1de-4f73-a051-
33d13d5413bd}" Rundown="true">
      </EventProvider>
      <EventProvider Name="Microsoft-Windows-Diagtrack" Guid="{43ac453b-97cd-4b51-4376-
db7c9bb963ac}" Rundown="true">
      </EventProvider>
      <EventProvider Name="Microsoft-Windows-Kernel-Process" Guid="{22FB2CD6-0E7B-422B-A0C7-
2FAD1FD0E716}" Rundown="true">
      </EventProvider>
      <EventProvider Name="Microsoft-Windows-DNS-Client" Guid="{1c95126e-7eea-49a9-a3fe-
a378b03ddb4d}" Rundown="true">
      </EventProvider>
      <EventProvider Name="Microsoft-Windows-Kernel-File" Guid="{edd08927-9cc4-4e65-b970-
c2560fb5c289}" Rundown="true">
      </EventProvider>
    </EventProviders>
  </Provider>
</SystemActivityMonitor>
```

SAMRecordingProfile.xsd

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="SystemActivityMonitor">
    <xs:complexType>
      <xs:all>
        <!-- Collector Level -->
        <xs:element minOccurs="0" name="Collector">
          <xs:complexType>
            <xs:all>
              <!-- SystemCollector -->
              <xs:element minOccurs="0" name="SystemCollector">
                <xs:complexType>
                  <xs:all>
                    <xs:element minOccurs="0" name="BufferSize">
                      <xs:complexType>
                        <xs:attribute name="Value" type="xs:integer" use="optional" />
                      </xs:complexType>
                    </xs:element>
                    <xs:element minOccurs="0" name="MinimumBuffers">
                      <xs:complexType>
                        <xs:attribute name="Value" type="xs:integer" use="optional" />
                      </xs:complexType>
                    </xs:element>
                    <xs:element minOccurs="0" name="MaximumBuffers">
                      <xs:complexType>
                        <xs:attribute name="Value" type="xs:integer" use="optional" />
                      </xs:complexType>
                    </xs:element>
                    <xs:element minOccurs="0" name="FlushTimer">
                      <xs:complexType>
                        <xs:attribute name="Value" type="xs:integer" use="optional" />
                      </xs:complexType>
                    </xs:element>
                  </xs:all>
                  <xs:attribute name="Name" type="xs:string" use="required" />
                </xs:complexType>
              </xs:element>
              <!-- EventCollector -->
              <xs:element minOccurs="0"  name="EventCollector">
                <xs:complexType>
                  <xs:all>
                    <xs:element minOccurs="0" name="BufferSize">
                      <xs:complexType>
                        <xs:attribute name="Value" type="xs:integer" use="optional" />
                      </xs:complexType>
                    </xs:element>
                    <xs:element minOccurs="0" name="MinimumBuffers">
                      <xs:complexType>
                        <xs:attribute name="Value" type="xs:integer" use="optional" />
                      </xs:complexType>
                    </xs:element>
                    <xs:element minOccurs="0" name="MaximumBuffers">
                      <xs:complexType>
                        <xs:attribute name="Value" type="xs:integer" use="optional" />
                      </xs:complexType>
                    </xs:element>
                    <xs:element minOccurs="0" name="FlushTimer">
                      <xs:complexType>
                        <xs:attribute name="Value" type="xs:integer" use="optional" />
                      </xs:complexType>
                    </xs:element>
                  </xs:all>
                  <xs:attribute name="Name" type="xs:string" use="required" />
                </xs:complexType>
              </xs:element>
              <!-- Filter -->
              <xs:element minOccurs="0" name="Filter">
                <xs:complexType>
                  <xs:sequence>
                    <xs:choice minOccurs="0" maxOccurs="unbounded">
                      <xs:element name="CommandLine">
                        <xs:complexType>
                          <xs:simpleContent>
                            <xs:extension base="xs:string">
                              <xs:attribute name="condition" type="xs:string" use="optional" />
                            </xs:extension>
                          </xs:simpleContent>
                        </xs:complexType>
                      </xs:element>
                      <xs:element name="ParentCommandLine">
                        <xs:complexType>
                          <xs:simpleContent>
                            <xs:extension base="xs:string">
                              <xs:attribute name="condition" type="xs:string" use="optional" />
                            </xs:extension>
                          </xs:simpleContent>
                        </xs:complexType>
                      </xs:element>
                      <xs:element name="ImageName">
                        <xs:complexType>
                          <xs:simpleContent>
                            <xs:extension base="xs:string">
                              <xs:attribute name="condition" type="xs:string" use="optional" />
                            </xs:extension>
                          </xs:simpleContent>
                        </xs:complexType>
                      </xs:element>
                      <xs:element name="ParentImageName">
                        <xs:complexType>
                          <xs:simpleContent>
                            <xs:extension base="xs:string">
                              <xs:attribute name="condition" type="xs:string" use="optional" />
                            </xs:extension>
                          </xs:simpleContent>
                        </xs:complexType>
                      </xs:element>
                    </xs:choice>
                  </xs:sequence>
                  <xs:attribute name="condition" type="xs:string" use="required" />
                </xs:complexType>
              </xs:element>
            </xs:all>
          </xs:complexType>
        </xs:element>
        <!-- Provider Level -->
        <xs:element minOccurs="0" name="Provider">
          <xs:complexType>
            <xs:all>
              <!-- SystemProviders -->
              <xs:element minOccurs="0" name="SystemProviders">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element minOccurs="0" maxOccurs="unbounded" name="SystemProvider">
                      <xs:complexType mixed="true">
                        <xs:sequence minOccurs="0" maxOccurs="1">
                          <xs:element name="Filter">
```

```
                              <xs:complexType>
                                <xs:choice minOccurs="0" maxOccurs="1">
                                  <xs:element minOccurs="0" maxOccurs="1" name="EventIdList"
type="intValuelist" />
                                  <xs:element minOccurs="0" maxOccurs="1" name="OpcodeIdList"
type="intValuelist" />
                                </xs:choice>
                                <xs:attribute name="condition" type="xs:string" use="required" />
                              </xs:complexType>
                            </xs:element>
                          </xs:sequence>
                          <xs:attribute name="Name" type="xs:string" use="required" />
                        </xs:complexType>
                      </xs:element>
                    </xs:sequence>
                  </xs:complexType>
                </xs:element>
                <!-- EventProviders -->
                <xs:element minOccurs="0" name="EventProviders">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element minOccurs="0" maxOccurs="unbounded" name="EventProvider">
                        <xs:complexType mixed="true">
                          <xs:sequence minOccurs="0" maxOccurs="1">
                            <xs:element name="Filter">
                              <xs:complexType>
                                <xs:choice minOccurs="0" maxOccurs="1">
                                  <xs:element minOccurs="0" maxOccurs="1" name="EventIdList"
type="intValuelist" />
                                  <xs:element minOccurs="0" maxOccurs="1" name="OpcodeIdList"
type="intValuelist" />
                                </xs:choice>
                                <xs:attribute name="condition" type="xs:string" use="required" />
                              </xs:complexType>
                            </xs:element>
                          </xs:sequence>
                          <xs:attribute name="Name" type="xs:string" use="optional" />
                          <xs:attribute name="Guid" type="xs:string" use="optional" />
                          <xs:attribute name="Level" type="xs:unsignedByte" use="optional" />
                          <xs:attribute name="Any" type="hexValue" use="optional" />
                          <xs:attribute name="All" type="hexValue" use="optional" />
                          <xs:attribute name="Rundown" type="xs:boolean" use="optional" />
                        </xs:complexType>
                      </xs:element>
                    </xs:sequence>
                  </xs:complexType>
                </xs:element>
              </xs:all>
            </xs:complexType>
          </xs:element>
        </xs:all>
        <!-- General Section -->
        <xs:attribute name="Author" type="xs:string" use="optional" />
        <xs:attribute name="Company" type="xs:string" use="optional" />
        <xs:attribute name="Team" type="xs:string" use="optional" />
        <xs:attribute name="Version" type="xs:decimal" use="optional" />
      </xs:complexType>
    </xs:element>
    <!-- Custom Types -->
    <!-- intValuelist -->
    <xs:simpleType name="intValuelist">
      <xs:list itemType="xs:integer"/>
    </xs:simpleType>
    <!-- hexValue -->
    <xs:simpleType name="hexValue">
      <xs:restriction base="xs:string">
        <xs:pattern value="0x[0-9A-Fa-f]+"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:schema>
```

Supported ETW Kernel Flags Names

```
Alpc: A provider that enables ALPC events //a provider that enables ALPC events
CSwitch: A provider that enables context switch events
DbgPrint: A provider that enables debug print events
DiskFileIO: A provider that enables file I/O name events
DiskIO: A provider that enables disk I/O completion events
DiskIOInit: A provider that enables disk I/O start events
FileIO: A provider that enables file I/O completion events
FileIOInit: A provider that enables file I/O start events
ReadyThread: A provider that enables thread dispatch events
DPC: A provider that enables device deferred procedure call events
Drivers: A provider that enables driver events
ImageLoader: A provider that enables image load events
Interrupt: A provider that enables interrupt events
HardPageFaultsOnly: A provider that enables memory hard fault events
AllPageFaults: A provider that enables memory page fault events
NetworkTrace: A provider that enables network tcp/ip events
Process: A provider that enables process events
ProcessCounter: A provider that enables process counter events
SampledProfile: A provider that enables profiling events
Registry: A provider that enables registry events
SplitIO: A provider that enables split I/O events
SystemCall: A provider that enables system call events
Thread: A provider that enables thread start and stop events
VAMap: A provider that enables file map and unmap (excluding images) events
VirtualAlloc: A provider that enables VirtualAlloc and VirtualFree events
ObjectManager: A provider that enables Object Manager events
```

Exemplary DiagTrack Recording Profile

```
<?xml version='1.0' encoding='utf-8' standalone='yes'?>
<SystemActivityMonitor Author="Dominik Phillips" Company="ERNW" Team="Windows Security"
Version="1.0">
  <Collector>
    <EventCollector Name="SAM_USER_SESSION">
      <BufferSize Value="1024"/>
      <MinimumBuffers Value="64"/>
      <MaximumBuffers Value="128"/>
    </EventCollector>
    <Filter condition="include">
      <CommandLine condition="contains">svchost.exe -k utcsvc -p</CommandLine>
      <ParentCommandLine condition="contains">svchost.exe -k utcsvc -p</ParentCommandLine>
    </Filter>
  </Collector>
  <Provider>
    <EventProviders>
      <EventProvider Name="Microsoft-Windows-Kernel-EventTracing" Guid="{b675ec37-bdb6-4648-bc92-
f3fdc74d3ca2}" Any="0x630" Rundown="true">
        <Filter condition="include">
          <EventIdList>2 3 8 9 12 14 15 19 26 27 29</EventIdList>
        </Filter>
      </EventProvider>
      <EventProvider Name="Microsoft-Windows-Kernel-Network" Guid="{7dd42a49-5329-4832-8dfd-
43d979153a88}" Rundown="true">
        <Filter condition="exclude">
          <EventIdList>42 43 49 58 59</EventIdList>
        </Filter>
      </EventProvider>
      <EventProvider Name="Microsoft-Windows-Kernel-Registry" Guid="{70eb4f03-c1de-4f73-a051-
33d13d5413bd}" Rundown="true">
        <Filter condition="include">
          <EventIdList>1 2 3 4 5 6 7 8 9</EventIdList>
        </Filter>
      </EventProvider>
      <EventProvider Name="Microsoft-Windows-Diagtrack" Guid="{43ac453b-97cd-4b51-4376-
db7c9bb963ac}" Rundown="true">
      </EventProvider>
      <EventProvider Name="Microsoft-Windows-Kernel-Process" Guid="{22FB2CD6-0E7B-422B-A0C7-
2FAD1FD0E716}" Any="0x70" Rundown="true">
        <Filter condition="include">
          <EventIdList>1 2 3 4 5 6</EventIdList>
        </Filter>
      </EventProvider>
      <EventProvider Name="Microsoft-Windows-DNS-Client" Guid="{1c95126e-7eea-49a9-a3fe-
a378b03ddb4d}" Rundown="true">
        <Filter condition="include">
          <EventIdList>3006 3008 3009 3020 3011 3020</EventIdList>
        </Filter>
      </EventProvider>
      <EventProvider Name="Microsoft-Windows-Kernel-File" Guid="{edd08927-9cc4-4e65-b970-
c2560fb5c289}" Rundown="true">
      </EventProvider>
    </EventProviders>
  </Provider>
</SystemActivityMonitor>
```

SAM Commands

-start

Starting an unmanaged recording session from a `cmd.exe` or `powershell.exe` session.

Syntax

```
sam.exe -start <recording profile id>
```

Description

`<recording profile id>`: Specifies the recording profile ID (i.e., the file of the SAM recording profile), that defines the properties of the recording session.

Examples

```
sam.exe -start 3
```

-add

Add a recording profile to the SAM's database

Syntax

```
sam.exe -add <recording profile file>
```

Description

`<recording profile file>`: Specifies the absolute file path of the SAM recording profile file, that should be added to the SAM database.

Examples

```
sam.exe -add C:\SAMProfiles\ExampleProfile.samp
```

-remove

Remove a recording profile from the SAM's database.

Syntax

```
sam.exe -remove <recording profile id>
```

Description

`<recording profile id>`: Specifies the recording profile ID (i.e. the file of the SAM recording profile), that should be removed from the SAM database.

Examples

```
sam.exe -remove 3
```

-list

List all available recording profiles stored in the SAM's database.

Syntax

```
sam.exe -list
```

Examples

```
sam.exe -list


list available recording profiles


   id   |               recording profile file
--------------------------------------------------------------------------
 1      | C:\recording_profiles\SAMContorlProfileExample.samp
```

-register_service

Register SAM as windows service with `<recording profile id>`

Syntax

```
sam.exe -register_service <recording profile id>
```

Description

`<recording profile id>`: Specifies the recording profile ID (i.e. the file of the SAM recording profile), that defines the properties of the recording session.

Examples

```
sam.exe -register_service 1
```

-unregister_service

Unregister SAM service

Syntax

```
sam.exe -ungregister_service
```

Examples

```
sam.exe -ungregister_service
```

-update_config

Update recording profile config of registered SAM service

Syntax

```
sam.exe -update_config <recording profile id>
```

Description

`<recording profile id>`: Specifies the recording profile ID (i.e. the file of the SAM recording profile), that defines the properties of the recording session.

Examples

```
sam.exe -update_config 3
```

-help | -h | -?

display usage information

Syntax

```
sam.exe -help
```

Examples

```
sam.exe -help
```

References

ERNW_WP4. (n.d.). SiSyPHuS Win10 (Studie zu Systemaufbau, Protokollierung, Härtung und Sicherheitsfunktionen in Windows 10): Work Package 4.

Keywords and Abbreviations