Federal Office
for Information Security

Microsoft Office Telemetry

Analysis report

Author:

Dr. Aleksandar Milenkoski

E-mail: amilenkoski@ernw.de

ERNW Enno Rey Netzwerke GmbH

Carl-Bosch-Straße 4

69115 Heidelberg, Germany

http://www.ernw.de

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

1 Introduction

ERNW GmbH was tasked by the German Federal Office for Information Security (orig., ger., Bundesamt fur Sicherheit in der Informationstechnik (BSI)) with analyzing the output of telemetry data from Microsoft Office and provide recommendations on how to deactivate or minimize it (see Section 1.1 and Section 1.2).

1.1 Concepts and Terms

This section introduces concepts and terms relevant for better understanding the content of this work.

The Windows 10 operating system implements the concept of telemetry. This involves collecting and sending diagnostic data to a backend managed by Microsoft, referred to as the *Microsoft backend,* for storing and processing. Diagnostic data is a set of diagnostic events that log information on different aspects of the operation of Windows and applications running on it. This includes usage information as well as information relevant for diagnosing issues, such as application crash information.

Connected User Experiences and Telemetry is the central telemetry component of Windows 10. The Connected User Experiences and Telemetry service, also known as DiagTrack, is the core building block of the component. It is responsible for collecting and sending diagnostic events to Microsoft.

DiagTrack primarily relies on Event Tracing for Windows (ETW) [ms_etw] for collecting diagnostic events. ETW is the core logging mechanism of Windows. The architecture of ETW is composed of several components, some of which are ETW providers, ETW sessions, and ETW consumers. An *ETW provider* is a software entity that produces log data (events). ETW providers are implemented and declared as part of instrumented executables using the ETW application programming interface (API) [ms_eapi]. Each ETW provider can be uniquely identified by a *globally unique identifier* (*GUID),* a 128-bit number. ETW providers deliver data to ETW sessions. An *ETW session* is a software entity that receives data from the ETW providers associated with it and delivers this data to ETW consumers. An *ETW consumer* is software that consumes and processes (e.g., displays) the data delivered by ETW sessions. For example, the Event Viewer utility is an ETW consumer. ([ERNW_WP4], Section 1.3) documents the architecture of ETW in greater detail.

The DiagTrack service receives diagnostic data from two ETW sessions, `Autologger-Diagtrack-Listener` and `Diagtrack-Listener`. `Autologger-Diagtrack-Listener` is active during system initialization. `Diagtrack-Listener` delivers diagnostic events to DiagTrack in the form of a real-time feed during system operation. [ERNW_WP4] discusses the architecture and operational principles of Connected User Experiences and Telemetry in greater detail.

It is important to emphasize that not only Connected User Experiences and Telemetry, but also applications developed by Microsoft may produce and send diagnostic events to the Microsoft backend. An example is Microsoft 365 (formerly called Microsoft Office 365), referred to as *Office* in this work. An Office deployment on a Windows instance consists of *Office applications*, such as Word and Excel, and the client-side implementations of *connected experiences*. Connected experiences are features of Office applications using services deployed in the Microsoft backend. They may communicate and exchange data with the Microsoft backend during operation. There are *non-deactivatable features* (Microsoft refers to them as *essential connected experiences)* and *deactivatable features* (Microsoft refers to them as *non-essential connected experiences*) [ms_cexp]. The non-deactivatable features cannot be deactivated by users. An example non-deactivatable feature is Licensing [ms_epriv]. It handles licensing of Office applications, such as verifying the validity of a deployed license. The deactivatable features may be deactivated by users through standard configuration interfaces, such as group policy settings or registry values. An example deactivatable feature is Insert Icons. It enables users to insert in documents icons provided by Microsoft online [ms_nepriv].

Both Office applications and connected experiences produce diagnostic events that may be sent to Microsoft. This work refers to this concept as *Office telemetry*. Office produces a specific diagnostic event when it performs a given activity, which may be triggered by users. Such activities include, for example, launching the Office application or saving a document. Diagnostic events produced by Office are sent to

Microsoft by telemetry modules. In this work, the term *telemetry module* refers to a software entity that establishes a network connection to an endpoint that is part of the Microsoft backend and sends diagnostic events produced by Office to it. There are Office and Windows telemetry modules. The term *Office telemetry module* refers to a telemetry module distributed with Office. The term *Windows telemetry module* refers to a telemetry module that is not distributed with Office (e.g., it is distributed with Windows).

Microsoft has released a set of privacy settings for Office, one of which enables users to configure the type and amount of diagnostic data that Office may send to the Microsoft backend. When deployed, it is available in the form of a group policy setting at the policy path `User Configuration\Administrative Templates\Microsoft Office 2016\Privacy\Trust Center\Configure the level of client software diagnostic data sent by Office to Microsoft`. It allows users to configure one of the following *diagnostic data levels*:

- `required`: this level configures Office to send to Microsoft the "***minimum data*** *needed to keep Office secure, up-to-date, and performing as expected on the device it's installed*" (cit., from the group policy setting description);
- `optional`: this level configures Office to send to Microsoft "***additional data*** *that helps make product improvements and provides enhanced information to help detect, diagnose, and remediate issues*" (cit., from the group policy setting description);
- `neither`: this level configures Office such that "***no diagnostic data*** *about Office client software running on the user's device is sent to Microsoft*" (cit., from the group policy setting description).

When the policy setting is not configured, the level `optional` is applied. For the sake of simplicity, this work refers to the policy setting `Configure the level of client software diagnostic data sent by Office to Microsoft` as `Office diagnostic level`. Configuring the diagnostic data level `required`, `optional`, or `neither` results in setting the registry value `HKEY_CURRENT_USER\Software\Policies\Microsoft\office\common\clienttelemetry\sendtelemetry` to 1, 2, and 3, respectively. This can be observed by analyzing the content of the `office16.admx` file. This file implements the `Office diagnostic level` setting (see Figure 1).

```
<policy name="L_SendTelemetry" class="User" displayName="$(string.L_SendTelemetry)" [...]
key="software\policies\microsoft\office\common\clienttelemetry">
  <parentCategory ref="L_TrustCenter" />
  <supportedOn ref="windows:SUPPORTED_Windows7" />
  <elements>
    <enum id="L_SendTelemetryDropID" valueName="sendtelemetry">
      <item displayName="$(string.L_SendTelemetryZero)">
        <value>
          <decimal value="3" />
        </value>
      </item>
      <item displayName="$(string.L_SendTelemetryFull)">
        <value>
          <decimal value="2" />
        </value>
      </item>
      <item displayName="$(string.L_SendTelemetryBasic)">
        <value>
          <decimal value="1" />
        </value>
      </item>
```

Figure 1: Content of office16.admx: Diagnostic data levels

1.2 Scope

The objective of this work is:

- to analyze the impact of the `required`, `optional`, and `neither` diagnostic data levels on the output of diagnostic data produced by Office applications and featured connected experiences (see Section 1.1). This work discusses in detail only topics that are directly related to this objective. Other topics are either not discussed, or are discussed to the extent needed for better understanding the content of this work. Such topics include characterizing the network traffic between a telemetry module and the Microsoft backend, or analyzing in detail the way in which Office applications and connected experiences produce diagnostic events;
- to provide and evaluate approaches for partially or fully disabling the output of diagnostic data produced by Office applications and featured connected experiences. This work evaluates such approaches in terms of their complexity of technical feasibility and impact on the operation of Office.

Office telemetry (see Section 1.1) is not to be confused with Office Telemetry Dashboard. Office Telemetry Dashboard is an on-premise tool that collects diagnostic data about Office, primarily intended for organization-internal application compatibility testing [ms_otd]. This data is different than the diagnostic data discussed in this work. The Office Telemetry Dashboard tool is not in the scope of this work.

The observations presented in this work are based on a static code analysis performed using the IDA disassembler and dynamic code analysis performed using the windbg debugger. Debugging symbols for Office are not publicly available.

1.2.1 Technical information

This section provides information on: *i)* the executable files that are subject of analysis and their execution environment (paragraph 'Operating system', 'Microsoft Office 365', and 'Executable files'); and *ii)* the configuration settings for Office that are subject of analysis (paragraph 'Configuration settings', see Section 1.1).

Operating system Microsoft Windows 10 Enterprise long-term servicing channel (LTSC), version 10.0.17763, build 17763, 64-bit, default configuration.

Microsoft Office 365 Microsoft Office 365 Business, version 1904, build 11601.20230 Click-to-Run, 64-bit, licensed, default configuration.

Installed Office applications (with all connected experiences enabled, as per the default configuration of Office):

- Access: `%ProgramFiles%\Microsoft Office\root\Office16\MSACCESS.EXE`
- Excel: `%ProgramFiles%\Microsoft Office\root\Office16\EXCEL.EXE`
- OneNote: `%ProgramFiles%\Microsoft Office\root\Office16\ONENOTE.EXE`
- Outlook: `%ProgramFiles%\Microsoft Office\root\Office16\OUTLOOK.EXE`
- PowerPoint: `%ProgramFiles%\Microsoft Office\root\Office16\POWERPNT.EXE`
- Publisher: `%ProgramFiles%\Microsoft Office\root\Office16\MSPUB.EXE`
- Word: `%ProgramFiles%\Microsoft Office\root\Office16\WINWORD.EXE`
- Skype for Business: `%ProgramFiles%\Microsoft Office\root\Office16\lync.exe`

Executable files

`%ProgramFiles%\Microsoft Office\root\vfs\ProgramFilesCommonX64\Microsoft Shared\OFFICE16\Mso20win32client.dll`: file version: 16.0.11601.20184; file size: 7.028.760 bytes.

`%ProgramFiles%\Microsoft Office\root\Office16\MSOARIANEXT.dll`: file version: 16.0.11601.20174; file size: 1.318.944 bytes.

Configuration settings: Administrative Template files (ADMX/ADML) and Office Customization Tool for Office 365 ProPlus, Office 2019, and Office 2016, version 4936.1000.

1.3 Summary

The Windows 10 operating system implements the concept of telemetry. This involves collecting and sending diagnostic data to a backend managed by Microsoft (i.e., the Microsoft backend) for storing and processing. Diagnostic data is a set of diagnostic events that log information on different aspects of the operation of Windows and applications running on it. This includes usage information as well as information relevant for diagnosing issues, such as application crash information. Microsoft Office 365 (i.e., Office) consists of Office applications, such as Word and Excel, and connected experiences. Connected experiences are features of Office applications that may communicate and exchange data with the Microsoft backend during operation. Both Office applications and connected experiences produce diagnostic events that may be sent to Microsoft. Microsoft has released a set of privacy settings for Office, in the form of group policy settings. One of them enables users to configure the type and amount of diagnostic data that Office may send to the Microsoft backend by configuring the diagnostic data level `required`, `optional`, or `neither`. According to the group policy setting description, the diagnostic data level `neither` configures Office such that "*no diagnostic data about Office client software running on the user's device is sent to Microsoft*" (cit., from the group policy setting description).

It is important to emphasize that:

- the diagnostic data level `neither` configures Office such that only specific diagnostic events are not sent to Microsoft. Other diagnostic events produced by Office applications and featured connected experiences are still sent to Microsoft;
- depending on how Office is used, diagnostic data produced by it may be sent to Microsoft through more than one telemetry module. Telemetry modules are software entities that collect Office diagnostic events, establish a network connection to an endpoint of the Microsoft backend, and send the diagnostic events to it. Office diagnostic data may be sent to Microsoft by telemetry modules distributed with Windows or Office itself.

  There is no known central configuration setting that disables all telemetry modules. There is also no such setting that configures Office to stop producing diagnostic events. Fully disabling the output of diagnostic data produced by Office requires the application of a combination of approaches. They involve blocking outgoing data streams at network-level and configuring settings using standard system configuration interfaces, such as group policy settings or the system's registry. The approaches vary in their efficacy (in terms of amount of disabled diagnostic data output), complexity of technical feasibility, and impact on the operation of Office applications and featured connected experiences. Partially or fully disabling the output of diagnostic data produced by Office limits Microsoft's ability to diagnose and remediate problems in using Office.

2 Technical Analysis

This section first provides a high-level overview of Office telemetry. It also discusses the impact of the `required`, `optional`, and `neither` diagnostic data levels on the output of diagnostic data produced by Office (Section 2.1 and Section 2.2).

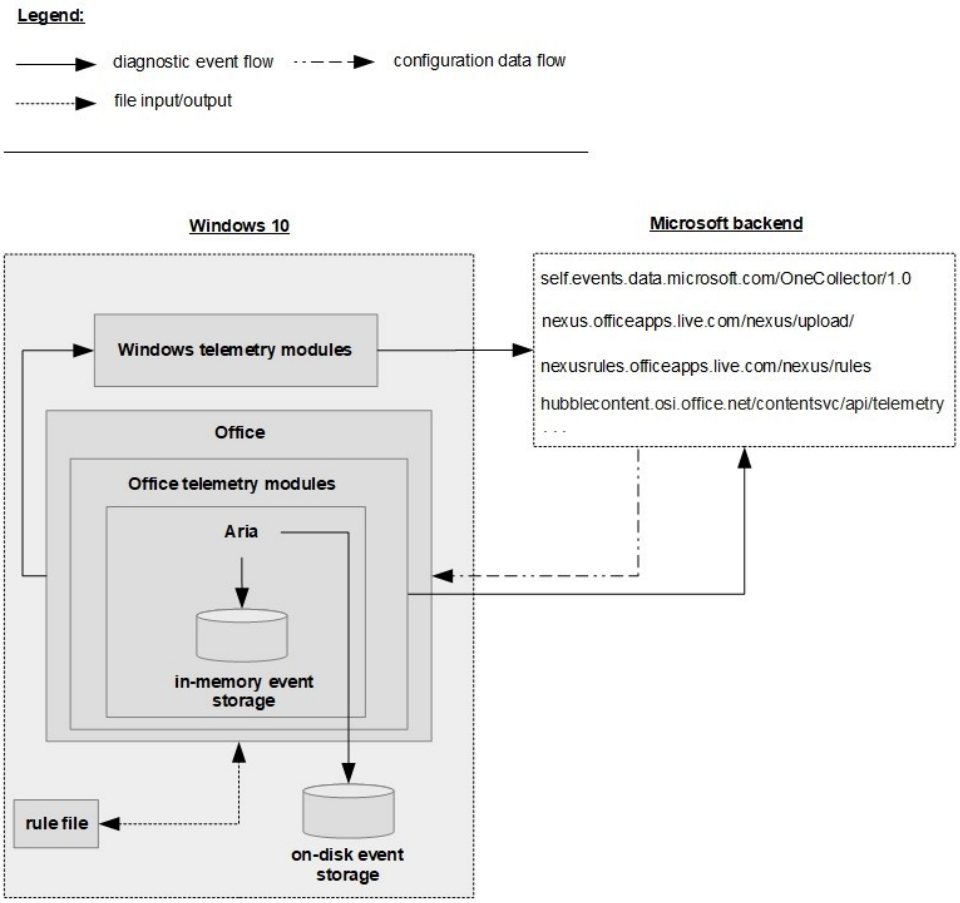

Figure 2: Office telemetry: A high-level overview (default configuration)

Figure 2 depicts a high-level overview of the architecture of Office telemetry. When an Office application or a featured connected experience is launched, it starts producing diagnostic events. These events may be sent to the Microsoft backend by Windows or Office telemetry modules ('Windows telemetry modules' and 'Office telemetry modules' in Figure 2, see Section 1.1).

Windows telemetry modules Some Windows telemetry modules are the Connected User Experiences and Telemetry service and the (Object Linking and Embedding) OLE32 Extensions for Win32 Windows component.

the Connected User Experiences and Telemetry service: The DiagTrack service, the core building block of Connected User Experiences and Telemetry, receives diagnostic events from the `Diagtrack-Listener` ETW session (see Section 1.1). An Office application may produce diagnostic events using ETW providers that are associated with `Diagtrack-Listener`. This means that these events are consumed by Connected User Experiences and Telemetry. Depending on its configuration, Connected User Experiences and Telemetry may send the events to Microsoft.

The section 'ETW Providers: Word and Diagtrack-Listener' in the Appendix lists the GUIDs of the ETW providers that: *i)* may be used by Word for producing diagnostic events; and *ii)* are associated with `Diagtrack-Listener` when Word runs. We identified the GUIDs of the ETW providers that may be used

by Word for producing diagnostic events with the `logman` utility: `logman query providers -pid [PID]`, where `[PID]` stands for process ID. We identified the GUIDs of the ETW providers associated with `Diagtrack-Listener` by executing the `Get-EtwTraceProvider -SessionName "Diagtrack-Listener"` PowerShell command. The section 'ETW Providers: Word and Diagtrack-Listener' in the Appendix lists the matching GUIDs. They may differ depending on system and Office state and configuration.

Figure 3 depicts the ETW provider with GUID `D1318FE0-16B7-4FB-b5F9-BA3CD54CD9CC` producing the diagnostic event named `Office.TelemetryEngine.FirstProcessed` (displayed with the `Message Analyzer` utility). This event is produced by a running Office application (Word) with a process ID (PID) of 7116. The ETW provider is associated with the `Diagtrack-Listener` ETW session (see section 'ETW Providers: Word and Diagtrack-Listener' in the Appendix). The event `Office.TelemetryEngine.FirstProcessed` is collected by Connected User Experiences and Telemetry when the telemetry level `Enhanced(Limited)` is configured [ms_el].

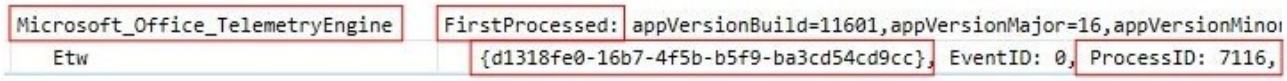

Figure 3: The diagnostic event Office.TelemetryEngine.FirstProcessed

OLE32 Extensions for Win32: When a user inserts an icon provided by Microsoft in a Word document (i.e., when a user uses the Insert Icon connected experience, see Section 1.1), Office uses the Windows OLE technology. This results in the `%SystemRoot%\urlmon.dll` Dynamic Link Library (DLL) library file sending diagnostic events to the `hubblecontent.osi.office.net/contentsvc/api/telemetry/ux?` endpoint of the Microsoft backend. `urlmon.dll` is part of the OLE32 Extensions for Win32 component. Figure 4 shows `urlmon.dll` constructing a POST request directed to `hubblecontent.osi.office.net/contentsvc/api/telemetry/ux?` (label [1]) using the WinINet library (label [2], [ms_wini]). The POST request encapsulates a diagnostic event in JavaScript Object Notation (JSON) format (label [3], displayed with Fiddler [fid]).

Office telemetry modules By default, Office uses the telemetry module implemented in the `%ProgramFiles%\Microsoft Office\root\Office16\MSOARIANEXT.dll` DLL library file. This work refers to this telemetry module as Aria ('Aria' in Figure 2). `MSOARIANEXT.dll` is loaded in the context of an Office application when launched. Aria is responsible for encapsulating diagnostic events produced by Office into POST requests and sending these requests to Microsoft. We observed that Aria sends diagnostic events to the `self.events.data.microsoft.com/OneCollector/1.0` endpoint (see Figure 2). Office delivers diagnostic events to Aria by invoking functions implemented in `MSOARIANEXT.dll`.

It is important to emphasize that there are multiple criteria based on which only specific diagnostic events produced by Office are actually delivered to Aria and sent to Microsoft. Section 2.2 discusses these criteria. This includes the impact of the `required`, `optional`, and `neither` diagnostic data levels on the output of diagnostic data produced by Office (see Section 1.2). Section 2.1 discusses the main functionalities of Aria.

If the registry value `HKEY_CURRENT_USER\Software\Policies\Microsoft\office\common\clienttelemetry\EnableWriteTelemetryEventsToNexus` is set to 1, Office applications do not use Aria, but the telemetry module referred to as Nexus in this work. We observed that this module sends diagnostic events to the `nexus.officeapps.live.com/nexus/upload` endpoint of the Microsoft backend (see Figure 2). A detailed analysis of Nexus is out of scope.

Figure 4: urlmon.dll constructing a POST request that encapsulates a diagnostic event

2.1 Functionalities of Aria

This section provides a high-level overview of the functionalities of the Aria telemetry module. The focus is on the functionalities that are relevant to the objectives of this work. Since debugging symbols are not available (see Section 1.2), this section refers to mentioned functions and variables using labels we assigned. Table 1 lists labels assigned to function and variables located at specific offsets from the base addresses of executables (i.e., images). In Table 1, in section 'Functions' and 'Variables': the column 'Label'/'Label [data type]' lists function and variable labels, and the type of the data stored in variables; the column 'Offset', sub-column 'Image', lists image names; the column 'Offset', sub-column 'Value', lists offsets from the base addresses of images.

By default, Office uses the Aria Office telemetry module for sending diagnostic events to the Microsoft backend. Office applications load the `MSOARIANEXT.dll` library file when launched, which is where Aria is implemented. Office then delivers diagnostic events to Aria by invoking functions implemented in `MSOARIANEXT.dll`. Aria stores these events, encapsulates them in POST requests, and sends them to Microsoft.

Functions		
Label	**Offset**	
	Image	**Value**
`aria_diagnosticEventLogger`	`MSOARIANEXT.dll`	`0x28A60`
`aria_offlineStorageTimeout`	`MSOARIANEXT.dll`	`0xD35DC`
Variables		
Label [data type]	**Offset**	
	Image	**Value**
`aria_logVerbosity` [byte]	`MSOARIANEXT.dll`	`0x12F9E0`

Table 1: Function and variable labels (1)

In order to observe the functionalities of Aria, we patched the value of the `aria_logVerbosity` variable to 0x4 when `MSOARIANEXT.dll` is loaded (see Table 1 and Figure 5). The value of this variable controls the verbosity level of the log data produced by Aria. The value 0x4 configures the highest verbosity level. Aria log data provides information on the inner working principles of Aria. It can be displayed by a debugger attached to an Office application that has loaded `MSOARIANEXT.dll` (see Figure 5).

```
[...]
start             end                 module name
00007ffa`19b80000 00007ffa`19cc1000   MsoAriaNext   (deferred)
    Image path: C:\Program Files\Microsoft Office\root\Office16\MsoAriaNext.dll
    Image name: MsoAriaNext.dll
[...]
0:011> ? 00007ffa`19b80000 + 0x12F9E0
Evaluate expression: 140712151284192 = 00007ffa`19caf9e0
0:011> eb 00007ffa`19caf9e0 0x4
0:011> ba w 1 00007ffa`19caf9e0
0:011> g
2020-03-25 14:17:09.689 T#4284 <D> [EventsSDK.AuthTokensController] New AuthTokensController instance
Breakpoint 0 hit
MsoAriaNext!MakeEventPropertiesImpl+0x26ed0:
00007ffa`19bb2b80 eb2e            jmp     MsoAriaNext!MakeEventPropertiesImpl+0x26f00 (00007ffa`19bb2bb0)
0:011> eb 00007ffa`19caf9e0 0x4
0:011> g
2020-03-25 14:17:16.835 T#4284 <D> [AriaSDK.PAL] Initializing...
[...]
2020-03-25 14:17:16.879 T#4284 <D> [EventsSDK.HttpClientFactory] Creating HttpClient_WinInet
2020-03-25 14:17:16.879 T#4284 <D> [EventsSDK.LogManager] BandwidthController: None
2020-03-25 14:17:16.888 T#4284 <D> [AriaSDK] resetStats start=1
2020-03-25 14:17:16.888 T#4284 <D> [EventsSDK.LogManager] Telemetry system created, starting up...
[...]
```

Figure 5: Patching aria_logVerbosity and displaying Aria log data

Once `MSOARIANEXT.dll` is loaded, Office starts delivering diagnostic events to Aria. The `aria_diagnosticEventLogger` function implemented in `MSOARIANEXT.dll` (see Table 1) is executed for each event delivered to Aria. Among other things, the execution of `aria_diagnosticEventLogger` results in storing the delivered event in an in-memory database that Aria manages ('in-memory event storage' in Figure 2). Figure 6 depicts Aria log data that provides information on the storage of the diagnostic event named `Office.Performance.Boot`.

```
[...]
0:010> ? 00007ffa`05550000 + 28a60
Evaluate expression: 140711808174688 = 00007ffa`05578a60
0:010> bp 00007ffa`05578a60
[...]
Breakpoint 0 hit
MsoAriaNext!MakeEventPropertiesImpl+0x1cdb0:
00007ffa`05578a60 488bc4          mov     rax,rsp
0:010> g
2020-03-25 14:34:14.958 T#3760 <D> [AriaSDK]
0000020528D5C690: LogEvent(properties.name="Office_Performance_Boot", ...)
2020-03-25 14:34:14.958 T#3760 <D> [AriaSDK]
Record=000000C5E9EFEED0 decorated with SemanticContext=000002052B439008
2020-03-25 14:34:14.958 T#3760 <D> [AriaSDK]
Record=000000C5E9EFEED0 decorated with SemanticContext=0000020528D5C7B0
2020-03-25 14:34:14.976 T#3760 <D> [AriaSDK]
Event 9b9b073d5a43495fae37f003b99e8ce3/custom.office_system_activity submitted,
priority 1 (Normal), serialized size 2251 bytes, ID ACF9B5B7-CFFB-4540-B7E7-F9F88E98ED20
[...]
```

Figure 6: Storage of a diagnostic event delivered to Aria

```
Breakpoint 3 hit
WININET!HttpOpenRequestA:                                              [1]
00007ffa`3a08cca0 488bc4          mov     rax,rsp
0:008> db @r8
00000027`66eff170  2f 4f 6e 65 43 6f 6c 6c-65 63 74 6f 72 2f 31 2e  /OneCollector/1.
00000027`66eff180  30 2f 00 66 27 00 00 00-b8 30 7f dc 5a 01 00 00  0/.f'....0..Z...
[..]
0:008> db @rdx
0000015a`dc734e38  50 4f 53 54 00 00 61 00-72 00 65 00 5c 00 43 00  POST..a.r.e.\.C.
[...]
Breakpoint 5 hit
WININET!HttpAddRequestHeadersA:
00007ffa`39f56530 4055            push    rbp
0:008> db @rdx L@r8
0000015a`da137e80  41 50 49 4b 65 79 3a 20-63 64 38 33 36 36 32 36  APIKey: cd836626
0000015a`da137e90  36 31 31 63 34 63 61 61-61 38 66 63 35 62 32 65  611c4caaa8fc5b2e
[...]
0000015a`da138060  70 3d 0d 0a 43 6c 69 65-6e 74 2d 49 64 3a 20 4e  p=..Client-Id: N
0000015a`da138070  4f 5f 41 55 54 48 0d 0a-43 6f 6e 74 65 6e 74 2d  O_AUTH..Content-
0000015a`da138080  45 6e 63 6f 64 69 6e 67-3a 20 64 65 66 6c 61 74  Encoding: deflat
0000015a`da138090  65 0d 0a 43 6f 6e 74 65-6e 74 2d 54 79 70 65 3a  e..Content-Type:
0000015a`da1380a0  20 61 70 70 6c 69 63 61-74 69 6f 6e 2f 62 6f 6e   application/bon
0000015a`da1380b0  64 2d 63 6f 6d 70 61 63-74 2d 62 69 6e 61 72 79  d-compact-binary
[...]
0:008> g
Breakpoint 1 hit
WININET!HttpSendRequestW:
00007ffa`39f55a70 48895c2410      mov     qword ptr [rsp+10h],rbx   [...]
```

```
                                                                       [2]
POST https://self.events.data.microsoft.com/OneCollector/1.0/ HTTP/1.1
[...]
Client-Id: NO_AUTH
Content-Type: application/bond-compact-binary
Expect: 100-continue
SDK-Version: EVT-Windows-C++-ECS-3.1.64.1
Upload-Time: 1585152611377
Host: self.events.data.microsoft.com
Content-Length: 4681
Connection: Keep-Alive
Cache-Control: no-cache

)□3.0I'Office_System_SystemHealthErrorsWithTagc   [...]
[...]
```

Figure 7: Aria constructing a POST request that encapsulates diagnostic events

If Internet connection is available, Aria schedules the sending of the diagnostic events stored in its in-memory database to the `self.events.data.microsoft.com/OneCollector/1.0` endpoint of the Microsoft backend. To this end, Aria executes functions of the WinINet library, such as `HttpOpenRequestA`, `HttpAddRequestHeadersA`, and `HttpSendRequestW` [ms_wini]. WinINet constructs POST requests such that each request encapsulates multiple diagnostic events. `HttpSendRequestW` issues POST requests to `self.events.data.microsoft.com/OneCollector/1.0` through an encrypted communication channel.

The events encapsulated in POST requests are serialized with the Bond Compact Binary protocol [ms_bond]. This protocol achieves high payload compactness. It is therefore suitable for scenarios where data has to be frequently sent over a network connection and the caused network overhead due to data transfer needs to be kept at minimum. Figure 7 depicts Aria constructing a POST request in order to send diagnostic events to `self.events.data.microsoft.com/OneCollector/1.0` (label [1]). The first event stored as part of this request is named `Office.System.SystemHealthErrorsWithTag` [ms_req]. Figure 7 also depicts the POST request as observed with Fiddler (label [2]). Fiddler acts as the man-in-the-middle between the Windows 10 instance where an Office application runs and the Microsoft backend.

When a user closes an Office application, if Internet connection is available, Aria sends to `self.events.data.microsoft.com/OneCollector/1.0` the diagnostic events that have not been already sent. If Internet connection is not available, Aria stores these diagnostic events in persistent storage ('on-disk event storage' in Figure 2). This storage is an SQLite database. It is stored in the `%HOMEPATH%\AppData\Local\Microsoft\Office\OTele\` folder, in a database (.db) file.[1] Aria stores diagnostic events in persistent storage during a timeout interval by invoking the `aria_offlineStorageTimeout` function (see Table 1). Windows terminates the Office application when the timeout expires.

2.2 Delivery of diagnostic events to Aria

This section discusses how diagnostic events produced by Office are delivered to Aria. The focus is on the criteria based on which only specific diagnostic events are delivered to Aria. Since debugging symbols are not available (see Section 1.2), this section refers to mentioned functions and variables using labels we assigned (see Table 2). Functions are located at specific offsets from the base addresses of images. Variables are located at offsets from the value of a parameter passed to a function, from a variable value, or from the base address of an image. As per Microsoft's function calling convention, the first parameter of a function receiving integers as parameters is stored in the `rcx` register, whereas the second in the `rdx` register [ms_cc]. In Table 2:

- in section 'Functions': the column 'Label' lists function labels; the column 'Offset', sub-column 'Image', lists image names; the column 'Offset', sub-column 'Value', lists offsets from the base addresses of images;
- in section 'Variables': the column 'Label [data type]' lists variable labels and the type of the data stored in variables; the column 'Offset', sub-column 'Image/Function/Variable', lists image names, function labels, or variable labels; the column 'Offset', sub-column 'Parameter', lists function parameters (where applicable); the column 'Offset', sub-column 'Value', lists offsets from the base addresses of images, from function parameter values, or from variable values.

1 `msaccess.exe.db` for Access; `excel.exe.db` for Excel; `onenote.exe.db` for OneNote; `outlook.exe.db` for Outlook; `powerpnt.exe.db` for PowerPoint; `mspub.exe.db` for Publisher; `winword.exe.db` for Word; `lync.exe.db` for Skype for Business.

Functions		
Label	**Offset**	
	Image	**Value**
`mso20_eventCheck`	`Mso20win32client.dll`	0x3D76C
`mso20_transportToAria2`	`Mso20win32client.dll`	0x331CC
`mso20_eventDLevelCheck`	`Mso20win32client.dll`	0x1C52F4
`mso20_transportToAria1`	`Mso20win32client.dll`	0x1523F0
`mso20_transportToAria`	`Mso20win32client.dll`	0x152610

Variables			
Label [data type]	**Offset**		
	Image/Function/Variable	**Parameter**	**Value**
`event_activation` [byte]	`mso20_eventCheck`	`rdx`	0x10
`event_diagnosticLevel` [byte]	`mso20_eventDLevelCheck`	`rcx`	0x6
`event_criticality` [byte]	`mso20_transportToAria1`	`rdx`	0x71
`setSettings` [pointer]	`mso20win32client.dll`	/	0x633200
`setTelemetryLevel` [byte]	`setSettings`	/	0x162

Table 2: Function and variable labels (2)

Figure 8 depicts the functions executed in close proximity to the `aria_diagnosticEventLogger` function, resulting in its execution (see Section 2.1). These functions are implemented in the `Mso20win32client.dll` DLL library file (see Section 1.2.1). Among other things, they evaluate properties of the diagnostic events directed to Aria. These properties serve as event descriptors and their values can be accessed as variables in the context of specific functions. The variables in Table 2 with names starting with `event_` store values of event properties. The event properties mentioned in this section are:

- `activation policy`: If set to 0x2, this property marks the event as deactivated. The value of this event property is stored in the `event_activation` variable (see Table 2);
- `diagnostic level`: This property is the diagnostic data level associated with the event (see Section 1.2). The value of this event property is stored in the `event_diagnosticLevel` variable (see Table 2). Possible values of `diagnostic level` are:
  - 0xA: marks the diagnostic data level `required` (internally named `B`);
  - 0x64: marks the diagnostic data level `optional` (internally named `F`);
  - 0x6E: marks the diagnostic data level internally named `N;`
  - 0x78: marks the diagnostic data level internally named `A`;[2]
- `criticality`: This property is the level of event criticality. The value of this event property is stored in the `event_criticality` variable (see Table 2). Possible values of `criticality` are:

2 We extracted the possible values of `diagnostic level` and their internal names from the context of the function implemented at offset 0x394520 from the base address of `Mso20win32client.dll`.

- 0x1: marks a non-critical event;
- 0xBF, 0xC0, 0xC1, and 0xC2: mark critical events of different criticalities (with internal names): 0XBF - `CriticalBusinessImpact`, 0xC0 – `CriticalCensus`, 0xC1 – `CriticalExperimentation`, and 0xC2 – `CriticalUsage`.[3]

The criteria based on which only specific diagnostic events are delivered to Aria take into account the values of these event properties. This section discusses only the `activation policy`, `diagnostic level`, and `criticality` event properties, since they are relevant to the objective of this work (see Section 1.2). There are other properties that are not discussed in this section.

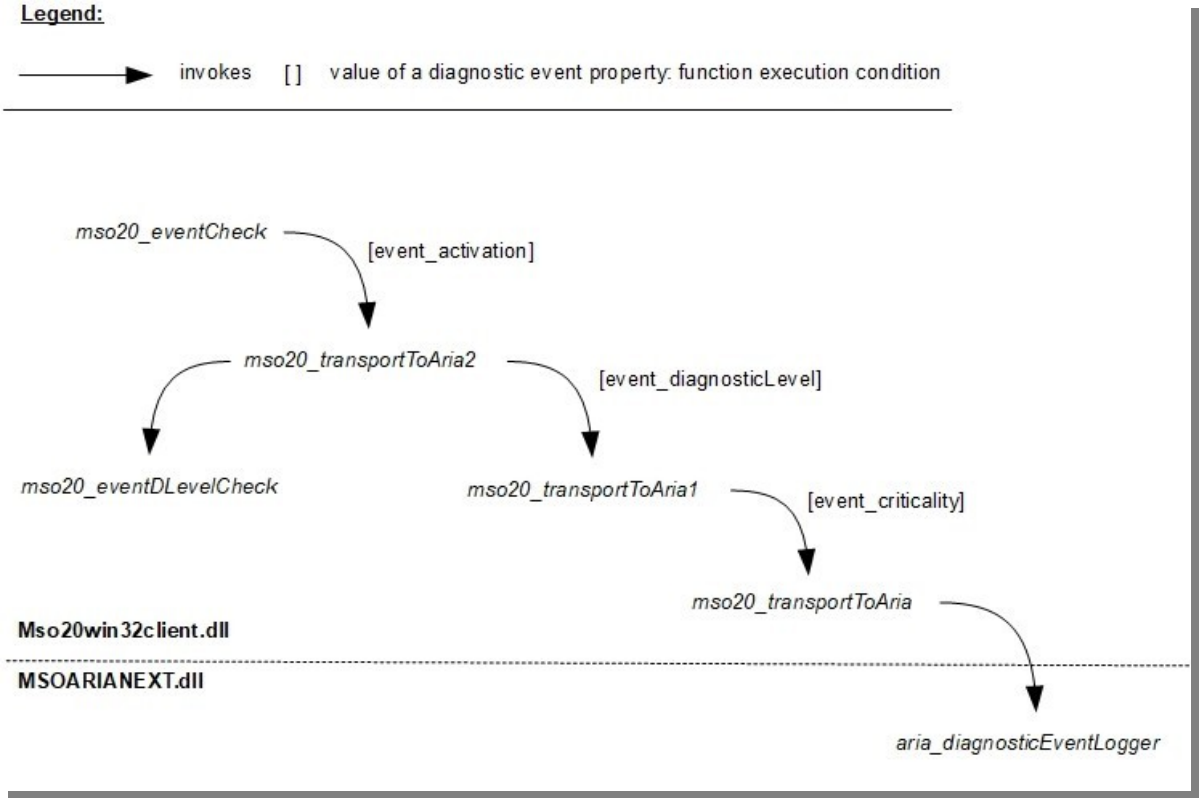

Figure 8: Functions executed in close proximity to aria_diagnosticEventLogger

For each diagnostic event directed to Aria, first the `mso20_eventCheck` function is invoked. This function evaluates different aspects of the event, such as whether the name of the event is valid or too long. It also evaluates the value of the `activation policy` event property. If an evaluation in `mso20_eventCheck` fails, the function returns a zero value and the function execution sequence depicted in Figure 8 is interrupted. The diagnostic event will not be delivered to Aria. `mso20_eventCheck` returns zero if the diagnostic event is marked as deactivated, that is, if the event property `activation policy` (or equivalently, the `event_activation` variable) is set to 0x2. Therefore, deactivated events are not delivered to Aria (see '[event_activation]' in Figure 8).

If `mso20_eventCheck` returns a non-zero value, the function `mso20_transportToAria2` is executed. This function invokes the function `mso20_eventDLevelCheck`. `mso20_eventDLevelCheck` evaluates the `diagnostic level` property of the event (or equivalently, the

3 We extracted the possible values of `criticality` and their internal names from the context of the function implemented at offset 0xB0660 from the base address of `Mso20win32client.dll` and by matching events of different criticalities sent to Microsoft with events specified in a rule file. Rule files are discussed later in this section.

event_diagnosticLevel variable) and returns 0 or 1. If it returns 0, the function execution sequence depicted in Figure 8 is interrupted. The diagnostic event will not be delivered to Aria.

Figure 9 depicts a pseudo-code of the implementation of mso20_eventDLevelCheck. This function decides whether a diagnostic event will be delivered to Aria by taking the diagnostic data levels required, optional, and neither into account. The setTelemetryLevel variable (see Table 2) stores the value of the registry value HKEY_CURRENT_USER\Software\Policies\Microsoft\office\common\clienttelemetry\sendtelemetry, which users set by configuring the Office diagnostic level policy setting. If event_diagnosticLevel is set to:

- 0x78 (i.e., A): mso20_eventDLevelCheck returns 1 and the diagnostic event is directed further to Aria;
- 0xA (i.e., required, or B): mso20_eventDLevelCheck returns 1 if the user has configured the policy setting Office diagnostic level to required (the registry value sendtelemetry is 1) or Optional (the registry value sendtelemetry is 2). The diagnostic event is directed further to Aria (see '[event_diagnosticLevel]' in Figure 8);
- 0x64 (i.e., optional, or F): mso20_eventDLevelCheck returns 1 if the user has configured the policy setting Office diagnostic level to optional (the registry value sendtelemetry is 2). The diagnostic event is directed further to Aria (see '[event_diagnosticLevel]' in Figure 8);
- 0x6E (i.e., N): mso20_eventDLevelCheck returns 1 if the value stored at offset 0x188 of the setSettings variable is set to 1. The diagnostic event is directed further to Aria. The role of the value stored at offset 0x188 of setSettings is not in the scope of this work.

In summary, a configured diagnostic data level required, optional, or neither is a criterion based on which a specific diagnostic event is delivered to Aria and sent to Microsoft only if the event's property diagnostic level is 0xA or 0x64.

```
mso20_eventLevelCheck()
{
    if (event_diagnosticLevel == 0x78) return 1;

    if (event_diagnosticLevel == 0xA)
    {
        if (setTelemetryLevel != 3) return 1;
        else return 0;
    }

    if (event_diagnosticLevel == 0x64)
    {
        if (setTelemetryLevel == 2) return 1;
        else return 0;
    }

    if (event_diagnosticLevel == 0x6E)
    {
        if (setSettings+0x188 == 1) return 1;
        else return 0;
    }
}
```

Figure 9: Pseudo-code of the implementation of mso20_eventDLevelCheck

If mso20_eventDLevelCheck returns 1, the function mso20_transportToAria1 is executed. Among other things, this function evaluates the event property criticality (or equivalently, the event_criticality variable). The value of this event property is relevant if the stream of diagnostic events is sampled. In such a case, only some diagnostic events are directed further to Aria and eventually

sent to Microsoft. Which events are directed further to Aria depends on the value of the criticality event property of each event. If the stream is sampled, `mso20_transportToAria1` delivers to Aria only the diagnostic events that are marked as critical – events with the `criticality` property set to `CriticalBusinessImpact`, `CriticalCensus`, `CriticalExperimentation`, or `CriticalUsage` (see '[event_criticality]' in Figure 8). To deliver a diagnostic event to Aria, `mso20_transportToAria1` invokes the `mso20_transportToAria` function. This function invokes the `aria_diagnosticEventLogger` function, at which point the diagnostic event is eventually sent to Microsoft (see Section 2.1).

The concept of event criticality is best observed in the context of telemetry rules. Telemetry rules are specified in Extensible Markup Language (XML) format and are stored in rule files ('rule file' in Figure 2). Rule files are stored in the `%HOMEPATH%\AppData\Local\Microsoft\Office\16.0\` folder, such that each Office application has a dedicated rule file.[4] An Office application may request a rule file from the `nexusrules.officeapps.live.com/nexus/rules` endpoint of the Microsoft backend (see Figure 10, label [1] marks a request for a rule file, label [2] marks a response from `nexusrules.officeapps.live.com`, displayed with Fiddler).

[1]

```
GET https://nexusrules.officeapps.live.com/nexus/rules?Application=winword.exe&Version=16.0.11601.20230&
ClientId=%7b19DBB312-0112-4FE9-8BCC-63EA6E392B74%7d&OSEnvironment=10&MsoAppId=0&AudienceName=Production&
AudienceGroup=Production&AppVersion=16.0.11601.20230& HTTP/1.1
Connection: Keep-Alive
Accept: application/vnd.ms-nexus-rules-v15+xml
Accept-Encoding: gzip
User-Agent: Microsoft Office/16.0 (Windows NT 10.0; Microsoft Word 16.0.11601; Pro)
X-MS-Collection-Policy: ExternalRestrictive, Heartbeat
X-MS-Process-Session-Id: {29B0E6CA-0E09-4345-B4E7-5A5C78340168}
Host: nexusrules.officeapps.live.com
```

[2]

```
HTTP/1.1 200 OK
Cache-Control: max-age=7200
Content-Length: 347202
Content-Type: application/vnd.ms-nexus-rules-v15+xml; charset=utf-8
[...]
<?xml version="1.0" encoding="utf-8"?>
<Rules xmlns="urn:Rules">
<R Id="1000" V="5" DC="ESM" EN="Office.Telemetry.RuleErrorsAggregated"
ATT="f998cc5ba4d448d6a1e8e913ff18be94-dd122e0a-fcf8-4dc5-9dbb-6afac5325183-7405"
SP="CriticalBusinessImpact" S="70" DL="A" DCa="PSP PSU" xmlns=""><S><Etw T="1" E="159"
G="{02fd33df-f746-4a10-93a0-2bc6273bc8e4}" />
<F T="2"><O T="AND"><L><O T="NE"><L><S T="1" F="Warning" /></L><R><V V="37" T="U32" /></R>
</O></L><R><O T="NE"><L><S T="1" F="Warning" /></L><R><V V="29" T="U32" /></R></O></R></O>
</F><TI T="3" I="10min" /><A T="4" E="TelemetrySuspend" /><A T="5" E="TelemetryShutdown" />
</S><G I="true" R="TriggerOldest">
[...]
```

Figure 10: An Office application (Word) requesting a rule file

Each rule stored in a rule file may contain sub-rules and is uniquely identified by a rule ID. Among other things, telemetry rules specify critical diagnostic events and event data sources. Some sources of diagnostic events are: ETW providers, uniquely identified by their GUIDs, and Unified Logging System (ULS) tags, uniquely identified by ULS tag IDs (e.g., `avuo1`, `bhyud`). ULS is an application logging mechanism. Each diagnostic event specified by a rule is associated with an event name and event criticality, also known as sampling policy. An Office application parses its rule file when started and matches over its lifetime

4 `msaccess.exe_Rules.xml` for Access; `excel.exe_Rules.xml` for Excel; `onenote.exe_Rules.xml` for OneNote; `outlook.exe_Rules.xml` for Outlook; `powerpnt.exe_Rules.xml` for PowerPoint; `mspub.exe_Rules.xml` for Publisher; `winword.exe_Rules.xml` for Word; `lync.exe_Rules.xml` for Skype for Business.

produced diagnostic events to rules. Matching events are directed to Aria. Therefore, rules may be understood as remotely deployed generators of diagnostic events dynamically configuring the diagnostic event collection process of Office (see 'configuration data flow' in Figure 2).

Figure 11 depicts portions of telemetry rules for Word. The rules specify events associated with the different event criticalities - `CriticalBusinessImpact`, `CriticalCensus`, `CriticalExperimentation`, and `CriticalUsage`. In Figure 11: the XML tag `R` specifies a rule; the XML attribute `EN` specifies an event name; the XML attribute `SP` specifies sampling policy (i.e., event criticality); the XML tag `Etw` specifies an ETW provider as an event data source; and the XML tag `UTS` specifies an ULS event data source.

```
<Rules xmlns="urn:Rules">
[...]
<R xmlns="" [...] SP="CriticalBusinessImpact" [...]
EN="Office.Telemetry.RuleErrorsAggregated" [..]>
    <S>
        <Etw G="{02fd33df-f746-4a10-93a0-2bc6273bc8e4}" E="159" T="1"/>
[...]
<R xmlns="" [...] SP="CriticalUsage" [...]
EN="Office.Outlook.Desktop.NdbCorruptionResult" [...]>
    <S>
        <Etw G="{2adf8e23-0af9-43c9-ba4c-952ee130540d}" E="368" T="1"/>
        </S>
[...]
<R xmlns="" [...] SP="CriticalExperimentation" [...]
EN="Office.Extensibility.DeepLinkingDocumentOpen" [...] >
    <S>
        <UTS Id="b072g" T="1"/>
    </S>
[...]
<R xmlns="" [...] SP="CriticalCensus" [...]
EN="Office.System.SystemHealthDesktopSessionLifecycleAndHeartbeat" [...] >
[...]
    <S>
        <UTS Id="awjb7" T="1"/>
        <UTS Id="a14x3" T="2"/>
[...]
```

Figure 11: Portions of telemetry rules

In addition to event names, criticalities, and event data sources, telemetry rules typically specify other event descriptors and operations over data, such as logical comparisons. A detailed analysis of telemetry rules is out of scope. To facilitate further research, the section 'Telemetry rules: XML tags/attributes and interpretations' in the Appendix non-exhaustively lists XML tags and attributes (column 'XML tag/attribute'), and their associated Office-internal interpretations (column 'Interpretations'). We extracted these interpretations as string literals from `Mso20win32client.dll`. Depending on its placement in a rule file, a single entry in the column 'XML tags/attribute' may be an XML tag or an attribute and may have more than one interpretation (comma-separated in column 'Interpretations').

Table 3 shows the number of diagnostic events directed and delivered to Aria, in the scenario where we conducted the following activities in Word: *i)* launching Word; *ii)* creating a new document; *iii)* writing the sentence "Test."; *iv)* saving the document using the 'Save as' feature; and *v)* closing Word. Under an event directed to Aria, we understand a diagnostic event reaching the `mso20_transportToAria1` function in `Mso20win32client.dll`. Under an event delivered to Aria, we understand a diagnostic event reaching the `mso20_transportToAria` function in `Mso20win32client.dll` and subsequently the `aria_diagnosticEventLogger` function in `MSOARIANEXT.dll` (see Figure 8).

For each diagnostic data level configured using the policy setting `Office diagnostic level`, Table 3 categorizes diagnostic events with respect to their criticality (column 'Event criticality') and diagnostic level (table section 'Event diagnostic level'). Section 'Total number of events delivered to Aria' of Table 3 shows the number of diagnostic events that are delivered to Aria and eventually sent to Microsoft (see Section 2.1).

Table 3 effectively shows the impact of configuring the policy setting `Office diagnostic level` on the output of diagnostic data produced by Office applications and featured connected experiences.

When the diagnostic data level `neither` is configured, events of the diagnostic levels 0xA and 0x64 are not directed to Aria. When the diagnostic data level `required` is configured, only events of the diagnostic level 0x64 are not directed to Aria (see Figure 8). When the diagnostic data level `optional` is configured, events of all diagnostic levels are directed to Aria, including a high number of events of the diagnostic level 0x64. Since the stream of diagnostic events is sampled, only the diagnostic events marked as critical (i.e., with the `criticality` event property set to `CriticalBusinessImpact`, `CriticalCensus`, `CriticalExperimentation`, or `CriticalUsage`) are delivered to Aria and eventually sent to Microsoft. This shows the impact of diagnostic event criticality on the output of diagnostic data produced by Office applications and featured connected experiences.

Event criticality	**Diagnostic data level**		
	`neither`	`required`	`optional`
Event diagnostic level: `0xA`			
Critical	0	10	10
Non-critical	0	1	1
Total	**0**	**11**	**11**
Event diagnostic level: 0x78			
Critical	20	19	21
Non-critical	8	8	8
Total	**28**	**27**	**29**
Event diagnostic level: `0x6E`			
Critical	2	2	2
Non-critical	28	29	28
Total	**30**	**31**	**30**
Event diagnostic level: `0x64`			
Critical	0	0	8
Non-critical	0	0	199
Total	**0**	**0**	**207**
Total number of events delivered to Aria			
	22	**31**	**41**

Table 3: Number of diagnostic events directed and delivered to Aria [the Word Office application with all connected experiences enabled, as per the default configuration of Office – see Section 1.2.1]

3 Disabling the output of diagnostic data

This section discusses different approaches for partially or fully disabling the output of diagnostic data produced by Office. It evaluates the approaches in terms of their complexity of technical feasibility and impact on the operation of Office. The approaches can be applied at network- (Section 3.1), registry- (Section 3.2), or group policy-level (Section 3.3). Although the approaches focus on diagnostic data, due to technical reasons, some of them involve disabling the output of non-diagnostic data required for non-essential connected experiences to function (see Section 1.1). The approaches discussed in this section do not disable the output of non-diagnostic data required for Office applications and non-deactivatable features, such as Licensing, to properly function.

This section focuses on approaches that can be applied using standard configuration interfaces, such as the system's registry or group policy settings. Other approaches, although technically feasible, are not discussed in this section. For example, the output of diagnostic data from the Aria telemetry module may be disabled by setting the second parameter of the `mso20_eventCheck` function (i.e., the `event_activation` variable) to 0x2 while an Office application runs. This configures the `activation policy` event property such that each diagnostic event directed to Aria is marked as deactivated (see Section 2.2).

The Connected User Experiences and Telemetry service may collect diagnostic data produced by Office and send it to Microsoft (see Section 2). [ERNW_WP4.1] discusses approaches on disabling the output of diagnostic data from this service. This telemetry module is not in the scope of this section.

3.1 Network

This approach involves blocking using a firewall outgoing diagnostic data streams from a Windows instance to endpoints of the Microsoft backend. This includes the endpoints to which Office and Windows telemetry modules send diagnostic events (see Section 1.1). In the context of this work, we observed that the Aria and Nexus Office telemetry modules send diagnostic events to `self.events.data.microsoft.com/OneCollector/1.0` and `nexus.officeapps.live.com/nexus/upload`. We also observed that OLE32 Extensions for Win32 sends diagnostic events to`hubblecontent.osi.office.net/contentsvc/api/telemetry/`. [ERNW_WP4.1] provides information on the endpoints to which the Connected User Experiences and Telemetry service sends diagnostic events.

If applied to known endpoints, this approach is easily technically feasible - it is implemented by configuring firewall rules. However, its sustainability and efficacy, in terms of amount of disabled diagnostic data output, are challenged by the following factors:

- software updates and re-configuration: The endpoints of the Microsoft backend to which telemetry modules send diagnostic data may change over time, for example, due to software updates re-configuring the modules. This challenges the sustainability of this approach;
- user activities: Office produces a specific diagnostic event when it performs a given activity, often triggered by users. Diagnostic events are delivered to telemetry modules, after which they send the events to endpoints of the Microsoft backend. For example, Aria sends to Microsoft diagnostic events produced when users perform a variety of activities, such as launching an Office application or saving a document (see Section 2.1). OLE32 Extensions for Win32 handles more specific diagnostic events. For example, it sends diagnostic events to Microsoft when a user uses the Insert Icon connected experience (see Section 2). To what specific endpoints diagnostic events produced by Office may be sent largely depends on performed user activities. To achieve full approach efficacy, these activities should be known before configuring firewall rules.

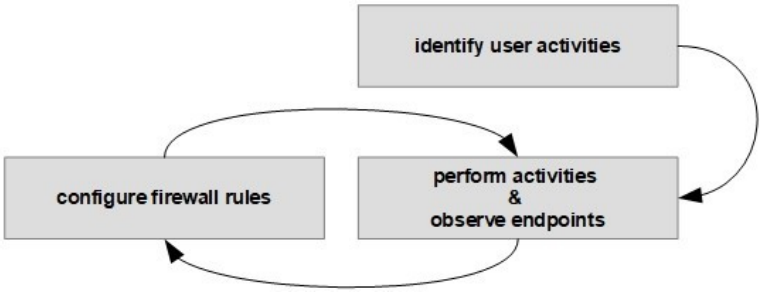

Figure 12: A sustainable and effective approach implementation

The sustainable and effective implementation of this approach involves performing the following steps:

1) identifying all activities that users may perform in the context of Office applications and featured connected experiences ('identify user activities' in Figure 12);

2) while performing the activities, observing the endpoints of the Microsoft backend to which diagnostic events are sent. This can be done by using a network sniffer acting as the man-in-the-middle between the Windows instance in which Office runs and the Microsoft backend (e.g., Fiddler, 'perform activities & observe endpoints' in Figure 12);

3) configuring firewall rules that block outgoing diagnostic data streams from the Windows instance to the observed endpoints ('configure firewall rules' in Figure 12). If an endpoint is in the form of a hostname and an Uniform Resource Locator (URL) path (e.g., `self.events.data.microsoft.com/OneCollector/1.0`), it is important that configured firewall rules specify the endpoint by its hostname and the full URL path. This is because:

- there may be multiple IP addresses associated with a single hostname. For example, `self.events.data.microsoft.com` is associated with multiple IP addresses. This makes blocking outgoing data streams only to a specific IP address, or a hostname, inefficient;
- some hostnames specify endpoints that not only collect diagnostic events, but also deliver content to Office. For example, in addition to collecting diagnostic events at the URL path `/contentsvc/api/telemetry/`, `hubblecontent.osi.office.net` delivers on request icon data when the Insert Icon connected experience is used. Blocking outgoing data streams only to the hostname `hubblecontent.osi.office.net`, without specifying an URL path, impacts the operation of the Insert Icon connected experience.

  Windows Defender Firewall, the Windows built-in firewall, does not support blocking outgoing data streams to endpoints specified by hostnames and URL paths. Therefore, the implementation of this approach requires a third-party firewall solution that supports such blocking. This feature is typical for enterprise network filtering solutions.

  An alternative, technically simpler, approach to blocking outgoing streams to specific endpoints is to block all outgoing data streams from an Office application. However, this may render Office applications and featured connected experiences unusable, disabling both non-deactivatable and deactivatable features. This includes verifying or changing a license [ms_lic], inserting icons, and deploying new, or using existing, application extensions (i.e., add-ins).

Given non-changing user activities, step 2) and 3) should be regularly repeated so that the implementation of the approach is sustainable over long-term periods.

3.2 Registry

Setting the registry value `HKEY_CURRENT_USER\Software\Policies\Microsoft\office\common\clienttelemetry\DisableTelemetry` to 1 disables the Aria and Nexus Office telemetry modules (see Section 2). For example, if `DisableTelemetry` is set to 1, Office applications do not load the `MSOARIANEXT.dll` library file, which implements Aria (see Section 2.1). The registry value `DisableTelemetry` cannot be configured through the privacy policy settings for Office that Microsoft has released (see Section 1.1). It has to be configured by editing the system's registry. `DisableTelemetry` is not present in the registry by default. Its default value, if a user does not explicitly create and configure `DisableTelemetry`, is 0. It is important to emphasize that the `DisableTelemetry` registry value is not officially documented by Microsoft. Therefore, its impact on the output of Office diagnostic data may be subject to change without public notice.

The advantage of this approach is its simple technical feasibility– it is implemented by configuring a registry value. It is also effective in disabling the output of diagnostic data from the Aria and Nexus telemetry modules. It also does not impact the operation of Office applications and featured connected experiences. However, it disables the output of diagnostic data only from the Aria and Nexus Office telemetry modules. It does not disable, for example, the output of diagnostic data produced by connected experiences, sent to Microsoft by Windows telemetry modules (see Section 2). For example, when `DisableTelemetry` is set to 1, OLE32 Extensions for Win32 still sends diagnostic events to `hubblecontent.osi.office.net/contentsvc/api/telemetry` when a user uses the Insert Icon connected experience in Word.

In addition to `DisableTelemetry`, there are other registry values that are relevant for disabling the output of diagnostic data produced by Office. They can be configured through group policy settings and are therefore discussed in Section 3.3.

3.3 Group policy

This approach involves configuring group policy settings. Configuring the policy setting:

- at the policy path `User Configuration\Administrative Templates\Microsoft Office 2016\Privacy\Trust Center\Configure the level of client software diagnostic data sent by Office to Microsoft` to `Enabled`;
  - and `Type of diagnostic data` to `Neither` partially disables the output of diagnostic data from Aria such that only events of diagnostic level 0x78 and 0x6E are sent to Microsoft (see Table 3);
- at the policy path `User Configuration\Administrative Templates\Microsoft Office 2016\Privacy\Trust Center\Allow the use of connected experiences in Office` to `Disabled` disables all non-essential connected experiences and therefore disables the output of diagnostic data produced by them. However, the disabled connected experiences are then not available to users. This does not include non-deactivatable features, such as Licensing. Users may enable or disable a certain group of non-essential connected experiences by configuring the policy settings at the policy paths [ms_pc]:
  - `User Configuration\Administrative Templates\Microsoft Office 2016\Privacy\Trust Center\Allow the use of connected experiences that analyze content`;
  - `User Configuration\Administrative Templates\Microsoft Office 2016\Privacy\Trust Center\Allow the use of connected experiences that download online content`; and

- User Configuration\Administrative Templates\Microsoft Office 2016\Privacy\Trust Center\Allow the use of additional optional connected experiences in Office.

Configuring the policy settings above results in the setting of registry values:

- Allow the use of connected experiences in Office sets the registry value HKEY_CURRENT_USER\Software\Policies\Microsoft\office\16.0\common\privacy\disconnectedstate (1 – Enabled; 2 – Disabled);
- Allow the use of connected experiences that analyze content sets the registry value HKEY_CURRENT_USER\Software\Policies\Microsoft\office\16.0\common\privacy\usercontentdisabled (1 – Enabled; 2 – Disabled);
- Allow the use of connected experiences that download online content sets the registry value HKEY_CURRENT_USER\Software\Policies\Microsoft\office\16.0\common\privacy\downloadcontentdisabled (1 – Enabled; 2 – Disabled);
- Allow the use of additional optional connected experiences in Office sets the registry value HKEY_CURRENT_USER\Software\Policies\Microsoft\office\16.0\common\privacy\controllerconnectedservicesenabled (1 – Enabled; 2 – Disabled).

The advantage of this approach is its simple technical feasibility – it is implemented by configuring policy settings. It is also effective in disabling the output of diagnostic data produced by non-essential connected experiences and partially disabling this output from the Aria telemetry module. However, non-essential connected experiences are not available to users. In addition, the output of diagnostic data from Aria is not fully, but only partially, disabled.

3.4 Summary

The approaches presented in Section 3.1, Section 3.2, and Section 3.3 vary in their efficacy (in terms of amount of disabled diagnostic data output), complexity of technical feasibility, and impact on the operation of Office applications and featured connected experiences. With the goal to maximize the amount of disabled diagnostic data output with a minimum technical complexity, users may disable the Aria and Nexus telemetry modules as well as all non-essential connected experiences by configuring registry values.

The section 'Disabling the output of diagnostic data: .reg file' in the Appendix presents the content of a Registration Entries (.reg) file. When applied, it disables the Aria and Nexus Office telemetry modules by setting the registry value DisableTelemetry to 1 (see Section 3.2). It also disables all non-essential connected experiences by setting the registry values controllerconnectedservicesenabled, downloadcontentdisabled, and usercontentdisabled to 2 (see Section 3.3). Users may modify the values of controllerconnectedservicesenabled, downloadcontentdisabled, and usercontentdisabled in the .reg file to enable only specific groups of connected experiences (see Section 3.3). In addition, the file sets the registry values HKEY_CURRENT_USER\Software\Policies\Microsoft\office\16.0\common\qmenable and HKEY_CURRENT_USER\Software\Policies\Microsoft\office\16.0\common\sendcustomerdata to 0. This is equivalent to configuring the policy settings User Configuration\Administrative Templates\Microsoft Office 2016\Privacy\Trust Center\Enable Customer Experience Improvement Program and User Configuration\Administrative Templates\Microsoft Office 2016\Privacy\Trust Center\Send personal information to Disabled.[5]

5 The CIS Microsoft Office 2016 Benchmark ([cis_of], Section 2.24.1) recommends configuring these policy settings to Disabled. The impact of the settings on the output of diagnostic data produced by Office is not in the scope of this work (see Section 1.2).

In addition to disabling Aria and Nexus as well as all non-essential connected experiences, users may block any remaining outgoing streams of diagnostic data at network-level (see Section 3.1). Users may also disable the output of Office diagnostic data from Connected User Experiences and Telemetry (see Section 2, [ERNW_WP4.1]). It is important to note that partially or fully disabling the output of diagnostic data produced by Office limits Microsoft's ability to diagnose and remediate problems in using Office [ms_req].

Appendix

ETW Providers: Word and Diagtrack-Listener

ETW provider GUIDs (total number: 52)	
30336ED4-E327-447C-9DE0-51B652C86108	7ACF487E-104B-533E-F68A-A7E9B0431EDB
03BBE5B8-C788-4D0B-B47E-5B5731398A89	7B1AE42D-B4F2-414D-9C97-913F19049964
05F95EFE-7F75-49C7-A994-60A55CC09571	7C29709D-3C02-47FB-8A39-D8287522FADB
0616F7DD-722A-4DF1-B87A-414FA870D8B7	7E32A1C4-D502-5B7C-39E8-2B7B0B5F0424
072665FB-8953-5A85-931D-D06AEAB3D109	86CC27EA-6F87-47F7-8B43-3473527D4A87
077B8C4A-E425-578D-F1AC-6FDF1220FF68	8CCCA27D-F1D8-4DDA-B5DD-339AEE937731
0BCA4784-8257-51A0-D9EC-24FE1FE4C90D	93112DE2-0AA3-4ED7-91E3-4264555220C1
18608E62-A628-49D9-8C02-55972E097D24	9CA335ED-C0A6-4B4D-B084-9C9B5143AFF0
1A211EE8-52DB-4AF0-BB66-FB8C9F20B0E2	A40B455C-253C-4311-AC6D-6E667EDCCEFC
1AFF6089-E863-4D36-BDFD-3581F07440BE	A6D3C9AC-9128-522A-495A-1821191173C2
1CBA82B8-2B26-4D68-8447-1A3B85805B6A	B1642597-285E-560A-7F60-7E02F5DA22C0
2F50C5D0-E25E-4F89-AB4A-31C63B518D7A	B6FD710B-F783-4B1C-AB9C-C68099DCC0C7
319DC449-ADA5-50F7-428E-957DB6791668	BC71577F-76E9-583A-ECD6-62D0250D900F
32980F26-C8F5-5767-6B26-635B3FA83C61	C7E09E2A-C663-5399-AF79-2FCCD321D19A
364E2BEB-6EFC-47DC-B8B1-49AAE1D83922	CA967C75-04BF-40B5-9A16-98B5F9332A92
3720DDA7-CAEA-4AF3-A138-375AAFC3F1D6	D0F1A5C6-FC43-48AE-99BF-EFB1C38BE9D1
3C302A2A-F195-4FED-BD7B-C91BA3F33879	DCB453DB-C652-48BE-A0F8-A64459D5162E
3C430B0A-397A-4E1D-9A83-9C388405C00C	EA289C62-8C36-4904-9726-15ECD282AED5
3C74AFB9-8D82-44E3-B52C-365DBF48382A	EBADF775-48AA-4BF3-8F8E-EC68D113C98E
4E7ADD1A-6945-435A-82B6-612688BA9F57	F0558438-F56A-5987-47DA-040CA75AEF05
540DC156-E9D6-42DC-A225-29794149A495	F168D2FA-5642-58BB-361E-127980C64A1B
673CF800-208A-5327-3F4B-2BE44A66627A	F25BCD2E-2690-55DC-3BC4-07B65B1B41C9
703FCC13-B66F-5868-DDD9-E2DB7F381FFB	F3A71A4B-6118-4257-8CCB-39A33BA059D4
7067398C-BAE7-4191-BF16-C436DE658BAF	F4576912-C358-4374-A354-9040D45ABCC1
77AB313B-93DA-4AA5-A20F-9339FF5AE1E3	FF32ADA1-5A4B-583C-889E-A3C027B201F5
785E3EA5-A921-427C-8EDB-0583D49C7636	D1318FE0-16B7-4F5B-B5F9-BA3CD54CD9CC

Telemetry rules: XML tags/attributes and interpretations

XML tag/attribute	Interpretations
ETW	ETW source
S	sources, source, stop after
R	rule source, resets, right, regex
TI	timer
T	triggers, token, out type
C	column, count
G	group, provider ID, guid
F	file, filter, float
ATT	Aria tenant token
TR	this rule
A	app event, aggregator
TO	timeout
ID	rule ID
V	rule version, constant
TH	threshold
SQ	sequence
SR	string regex
EPS	ETW provider source
UTS	ULS tag source
SS	state source
UCSS	ULS category severity source
UACS	ULS all categories source
US	union source
ER	enable rules
E	enabled
EN	event name
RIS	rule interfaces

ST	starts
DL	diagnostic level
DC	data classifications
SP	sampling policy
Dca	data categories
DR	disable rules
L	left
W	wstring
I8/16/32/64	INT8/16/32/64
U8/16/32/64	UINT8/16/32/64
D	double
B	bool
BIN	binary
FT	filetime
U	unary operator
O	operator, nullable
I	interval, index
N	name

Disabling the output of diagnostic data: .reg file

```
Windows Registry Editor Version 5.00

[HKEY_CURRENT_USER\Software\Policies\Microsoft\office\common\clienttelemetry]
"DisableTelemetry"=dword:00000001

[HKEY_CURRENT_USER\Software\Policies\Microsoft\office\16.0\common]
"sendcustomerdata"=dword:00000000
"qmenable"=dword:00000000

[HKEY_CURRENT_USER\Software\Policies\Microsoft\office\16.0\common\privacy]
"usercontentdisabled"=dword:00000002
"downloadcontentdisabled"=dword:00000002
"controllerconnectedservicesenabled"=dword:00000002
```

Reference Documentation

ms_etw	https://docs.microsoft.com/en-us/windows/win32/etw/about-event-tracing [Retrieved: 09/04/2020]
ms_eapi	https://docs.microsoft.com/en-us/windows/win32/etw/event-tracing-reference [Retrieved: 09/04/2020]
ERNW_WP4	ERNW GmbH: SiSyPHuS Win10 (Studie zu Systemaufbau, Protokollierung, Härtung und Sicherheitsfunktionen in Windows 10): Work Package 4
ms_cexp	https://docs.microsoft.com/en-us/deployoffice/privacy/connected-experiences [Retrieved: 09/04/2020]
ms_epriv	https://docs.microsoft.com/en-us/deployoffice/privacy/essential-services [Retrieved: 09/04/2020]
ms_nepriv	https://docs.microsoft.com/en-us/deployoffice/privacy/connected-experiences [Retrieved: 09/04/2020]
ms_otd	https://docs.microsoft.com/en-us/deployoffice/compat/compatibility-and-telemetry-in-office [Retrieved: 09/04/2020]
ms_el	https://docs.microsoft.com/en-us/windows/privacy/enhanced-diagnostic-data-windows-analytics-events-and-fields [Retrieved: 09/04/2020]
ms_wini	https://docs.microsoft.com/en-us/windows/win32/wininet/portal [Retrieved: 09/04/2020]
fid	https://www.telerik.com/fiddler [Retrieved: 09/04/2020]
ms_bond	https://github.com/microsoft/bond [Retrieved: 09/04/2020]
ms_req	https://docs.microsoft.com/en-us/deployoffice/privacy/required-diagnostic-data [Retrieved: 09/04/2020]
ms_cc	https://docs.microsoft.com/en-us/cpp/build/x64-calling-convention?view=vs-2019 [Retrieved: 09/04/2020]
ERNW_WP4.1	ERNW GmbH: SiSyPHuS Win10 (Studie zu Systemaufbau, Protokollierung, Härtung und Sicherheitsfunktionen in Windows 10): Work Package 4.1
ms_lic	https://docs.microsoft.com/en-us/deployoffice/overview-licensing-activation-microsoft-365-apps [Retrieved: 09/04/2020]
ms_pc	https://docs.microsoft.com/en-us/deployoffice/privacy/manage-privacy-controls [Retrieved: 09/04/2020]
cis_of	Center for Internet Security: CIS Microsoft Office 2016 Benchmark

Keywords and Abbreviations