Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Studying the logging capability of Windows Telemetry component using Reverse Engineering

Tesis de Licenciatura en Ciencias de la Computación

Pablo Agustín Artuso
LU: 282/11
artusopablo@gmail.com

Director: Rodolfo Baader <rbaader@dc.uba.ar>
Codirector: Aleksandar Milenkoski <amilenkoski@ernw.de>

Buenos Aires, 2019

ABSTRACT (ENGLISH VERSION)

Windows, one of the most popular OS, has a component called Telemetry. It collects information from the system with the goal of analyzing and fixing software & hardware problems, improving the user experience, among others. The kind of information that can be obtained by this component is partially configurable in four different levels: security, basic, enhanced and full, being "security" the level where less information is gathered and "full" the opposite case.

How Telemetry stores/process/administrates the information extracted? It employs a widely used framework called Event Tracer for Windows (ETW) [4]. Embedded not only in userland application but also in the kernel modules, the ETW framework has the goal of providing a common interface to log events and therefore help to debug and log system operations, by instrumenting it.

In this work, we are going to analyze a part of the Windows kernel to better understand how Telemetry works from an internal perspective. This work will make windows analysts, IT admins or even windows users, more aware about the functionality of the Telemetry component helping to deal with privacy issues, bug fixing, knowledge of collected data, etc. Our analysis implies performing reverse engineering [5],[6] on the Telemetry component, which involves chal- lenging processes such as kernel debugging, dealing with undocumented kernel internal structures, reversing of bigger frameworks (i.e: ETW), binary libraries which lack symbols, etc. As part of the analysis we will also develop an in depth comparison between the differences among each level of Telemetry, stressing the contrast in the amount of events written, verbosity of information, etc. Finally, we will study how the communication between the Windows instance and the Microsoft backend servers is carried out.

ABSTRACT (SPANISH VERSION)

ACKNOWLEDGMENTS

CONTENTS

1. Motivation . . . 6

2. Introduction . . . 7
   - 2.1 Basic concepts . . . 7
     - 2.1.1 Reverse Engineering . . . 7
     - 2.1.2 Debugging . . . 7
   - 2.2 Tools . . . 7
     - 2.2.1 IDA pro . . . 7
     - 2.2.2 WinDBG . . . 7
     - 2.2.3 YARA . . . 8
     - 2.2.4 XPERF? . . . 8
   - 2.3 Windows components . . . 8
     - 2.3.0.1 Event Tracing for Windows . . . 8
     - 2.3.0.2 Telemetry . . . 8

3. Previous Work . . . 9

4. Analysis . . . 10
   - 4.1 Understanding how Telemetry makes use of ETW . . . 10
     - 4.1.0.1 Reversing registration process . . . 11
   - 4.2 When and how providers are registered . . . 12
   - 4.3 How writes are carried out . . . 12
   - 4.4 Relation between ETW session and ETW providers . . . 12
   - 4.5 Identifying the buffers . . . 12
   - 4.6 Provider GUID vs Group Provider GUID . . . 12
   - 4.7 Checking correctness of logged events . . . 12
   - 4.8 Automatization of event logging . . . 12
   - 4.9 Service isolation . . . 12
   - 4.10 Triggers . . . 12
   - 4.11 searching for new triggers . . . 12
   - 4.12 Difference among configuration levels of telemtry . . . 12
   - 4.13 Analysis of sent data over the channel to Microsfot backend services . . . 12

5. Results . . . 13

6. Conclusions . . . 14

7. References . . . 15

1. MOTIVATION

The following analysis was performed against the Telemetry component of the Windows OS, with looking forward to achieve the following goals:

- Understand how the process of generating logs was carried out.
- How, where and what logs were stored.
- Which applications are involved in gathering Telemetry information

2. INTRODUCTION

The analysis presented, was carried out in a particular version of the Windows OS. Specifically, Windows 10 64 bits Enterprise, 1607. Windows delivers a new version, usually, each .. Some words about ERNW and the project with the FBI

There are several reasons why this version was chosen:

- It was one of the mainstream version of Windows at the moment of starting the project.
- It was a long support version (EOS: April 2019).
- It was used by the German Police Office

2.1 Basic concepts

This section will describe basic concepts needed to fully understand the carried out process to perform the analysis.

2.1.1 Reverse Engineering

Software engineering can be defined as the process of designing, building and testing computer software. The processor of a computer, works with 1's and 0's, therefore developing any kind of sofware will mean

Key concepts about what RE means, from a general perspective, how it should tackled which tools are usually there. 64 bits, calling convention ..

2.1.2 Debugging

Explaniation of the concept of debugging, what is the different with doing static RE. Some words specifically for KERNEL debugging.

2.2 Tools

2.2.1 IDA pro

Introduction to IDA pro. Explaniation of what it is and how it works.

2.2.2 WinDBG

Introduction to WINDBG. Explaniation of what it is and how it works.

2.2.3 YARA

2.2.4 XPERF?

2.3 Windows components

2.3.0.1 Event Tracing for Windows

Complete explainiation of how it works due to its importance for the rest of the analysis. Different components: session, providers, consumers ,etc . Talk about the guid of the providers

2.3.0.2 Telemetry

Full description of the different features / characteristic which are involved in this analysis. Explaniation of how the worflow of the data is followed. talk about the name of the session,the levels of configuration, when it can be configured.. etc.

3. PREVIOUS WORK

Some lines about previous works in this topic. Most of them focused on just analysis from traffic / documentation.

4. ANALYSIS

Several files (dynamic libraries, executables, drivers) were analyzed and reversed in order to achieve the aforementioned goals. However, the main analysis was performed in the **ntoskrnl.exe** file, which is the one holding the actual implementation of the Windows Kernel.

It's important to stress that most of the general analysis was carried out using the Basic level of Telemetry.

Further sections will depict different challenges and achievements faced during the analysis.

4.1 Understanding how Telemetry makes use of ETW

Where, who and what were some of the questions that needed to be answered in order to be able to get information about the providers that were registered against the DiagTrack session. It's possible to obtain the whole list of providers registered to a particular session, by executing the following powershell command:

```
Get-EtwTraceProvider | where {$_.SessionName -match "<SESSION_NAME>"}
```

Fig. 4.1: Powershell comand to list ETW providers registered against a particular session.

One way to answer all the aforementioned questions, was to intercept the moment when some provider was going to write a message. If a breakpoint was set at that exact moment, it would be possible to gather information such as:

1. The piece of code that triggered the write (by inspecting the call stack of the function).
2. The actual content of the log being written.

However, the function being used for executing writes (**EtwWrite**) was not just used by the providers attached to the DiagTrack session but also by all the providers using the ETW framework. It was necessary to find a way to filter them and only intercept the ones important for the analysis.

Using **??**, it was possible to extract a GUID's list of the providers that were attached to the DiagTrack session. With this information, it would be possible to make the breakpoint to be triggered only when the provider's GUID that was trying to write was in the list.

Nevertheless, this strategy had one minor issue. The function (**EtwWrite**) had five parameters and none of them would show directly the GUID:

```
C++
NTSTATUS EtwWrite(
  REGHANDLE              RegHandle,
  PCEVENT_DESCRIPTOR     EventDescriptor,
  LPCGUID                ActivityId,
  ULONG                  UserDataCount,
  PEVENT_DATA_DESCRIPTOR UserData
);
```

Fig. 4.2: Documentation for EtwWrite function [1].

The first parameter is the registration handler. This object is returned once the provider executed the registration (**EtwRegister)** successfully. On the other hand, the **EtwRegister** receives the GUID as parameter:

```
C++
NTSTATUS EtwRegister(
  LPCGUID            ProviderId,
  PETWENABLECALLBACK EnableCallback,
  PVOID              CallbackContext,
  PREGHANDLE         RegHandle
);
```

Fig. 4.3: Documentation for EtwRegister function [2].

Therefore, in order to perform the cross-check it was not enough with information from the **EtwWrite** function, but also information from the **EtwRegister** was needed. To summarize, to understand if the write was being done by a provider registered against the DiagTrack session it was necessary to:

1. Extract the whole list of providers registered attached to the DiagTrack session.
2. Intercept all the **EtwRegister** executions and check if the GUID being used was inside the list.
3. If it was, save the handler.
4. Intercept all the **EtwWrite** executions and check if the handler being used is one of the stored handlers.
5. If it was, the provider that is writing, is attached to the DiagTrack session.

Even though the strategy seemed to be theoretically promising, it was necessary to understand how to actually carry out these actions. Further sections will depict that process.

4.1.0.1 Reversing registration process

1. We couldn't ensure that the data was being written was actually going to the DiagTrack session .

4.2 When and how providers are registered

4.3 How writes are carried out

4.4 Relation between ETW session and ETW providers

4.5 Identifying the buffers

4.6 Provider GUID vs Group Provider GUID

4.7 Checking correctness of logged events

4.8 Automatization of event logging

4.9 Service isolation

4.10 Triggers

4.11 searching for new triggers

YARA

4.12 Difference among configuration levels of telemtry

4.13 Analysis of sent data over the channel to Microsfot backend services

5. RESULTS

6. CONCLUSIONS

GDPR?

7. REFERENCES

BIBLIOGRAPHY

[1] Bolin Ding, Janardhan Kulkarni and Sergey Yekhanin. Collecting Telemetry Data Privately. In proceedings of Neural Information Processing Systems Conference (NIPS), 2017.

[2] Vasyl Pihur, Úlfar Erlingsson and Aleksandra Korolova. RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response. In CCS, pages 1054–1067, 2014

[3] Microsoft Corporation. Dynamic collection analysis and reporting of telemetry data. US 9,590,880 B2, 2017.

[4] Tarik Soulami. Inside Windows debugging. Chapter 12, 2012

[5] Hausi A. Miiller, Jens H. Jahnke, Kenny Wong ,Dennis B. Smith , Scott R. Tilley , Margaret-Anne Storey. Reverse Engineering: A Roadmap. In Proceedings of the Conference on The Future of Software Engineering, Pages 47-60, 2000.

[6] Bruce Dang, Alexandre Gazet, Sbastien Josse, Elias Bachaalany. Practical Reverse Engineering: x86, x64, ARM, Windows Kernel, Reversing Tools, and Obfuscation. 2014.