Tracing and logging

Add chapter introduction

1.1 Event Tracing for Windows (ETW)

Figure out if components should be emphasized using emph

Event Tracing for Windows (ETW) is a logging mechanism that is built into the kernel of Windows. It is used by kernel-mode drivers and applications to provide realtime events and tracing features. While ETW is built into most drivers and applications made by Windows, it is also available for developers to use in their own applications. As most privileged applications built into Windows utilize ETW, it is a very good source for telemetry data related to discovering exploit attempts.

In the architecture of ETW events are at the centerpiece where they are created, managed and consumed by different event components[2]. These differentiate between event *providers*, event *consumers*, and event *controllers*. All of these event components handle the workflow of ETW, either by reading or writing, or by controlling the events in some way. This is demonstrated on Figure 1.1 (ETW model diagram[2]), where *sessions* are at the center of the ETW model. These sessions are controlled by an *controller* and hereafter consumed by a consumer. The following sections will go into detail of how each component works together to provide realtime tracing events.

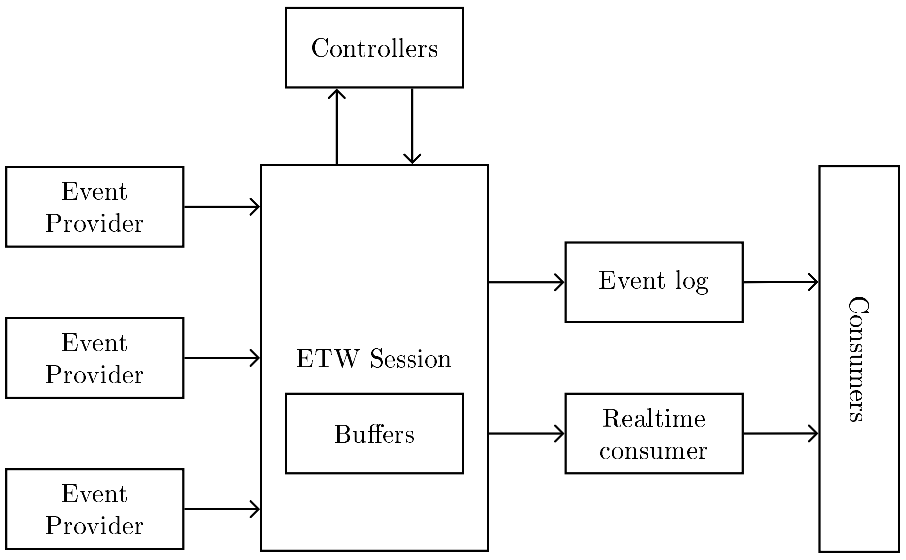


Figure 1.1: ETW model diagram[2]

1.1.1 Controllers

Controllers are applications, either user-mode or kernel-mode, used to start and manage trace sessions. ETW is special in the way that events are not stored or consumed in any way before a session is started. To start such a session, a *controller* application starts a trace using a Windows Application Programming Interface (API)s such as `StartTrace`. Afterwards specific *providers* can be enabled by using `EnableTraceEx2`. The specific API depend on the type of the provider as explained in the next section, 1.1.2. Controllers also manage buffers and statistics for events consumed in the current session.

1.1.2 Providers

Providers are the system- and userland applications that provide events and data. They do so by registering themselves as a provider, allowing a *controller* to enable or disable events. By having the *controller* control whether events are enabled or not, allows an application to have tracing without generating alerts all the time. This is especially interesting for debugging purposes, which is usually not needed during regular usage of the operating system.

Microsoft define four different types of providers depending on the version of Windows and type of application you are interested in. The reason for having four different types of providers is simply that ETW evolved over time, and as such different providers were added in different versions of Windows[8].

Managed Object Format (MOF) (classic) providers These types of providers are, as the name hints, the original format for specifying ETW providers. MOF providers use MOF classes[4] to define events. MOF classes describe the format of the event registered by the provider to allow the consumer to read the event correctly. As it can be seen on listing 1, a MOF class resemble a struct as known from the C programming language.

```
[EventType{26}, EventTypeName{"SendIPV6"}]
class TcpIp_SendIPV6 : TcpIp
{
    uint32 PID;
    uint32 size;
    object daddr;
    object saddr;
    object dport;
    object sport;
    uint32 startime;
    uint32 endtime;
    uint32 seqnum;
    uint32 connid;
};
```

Listing 1: `TcpIp_SendIPV6 : TcpIp` MOF class

Windows softwarre trace preprocessor (WPP) providers

With WPP providers, Windows moved away from using MOF classes to the Trace Message Format (TMF) format. With TMF the trace format description was moved into the Program Database (PDB) of the binary. For most binaries the PDB can be downloaded from Microsoft symbol servers[7], however not all Windows drivers and applications have public debug symbols, so getting access to the TMF is often a hit or miss.

Manifest-based providers

With manifest-based providers a new format to describe events was implemented. Instead of embedding the format description into the PDB, manifest-based providers embed the manifest directly into the binary as pointed to by the registry keys under `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\WINEVT\Publishers`. However, the manifest format is not well documented, making it hard parse and recover the schema needed to understand the events[1]. Manifest-based provider are however the first ETW provider type with the ability to be enabled by more than one trace session simultaneously, which is not possible with MOF and WPP providers.

TraceLogging providers

These types of providers are the newest type of providers in the ETW logging mechanism. Unlike all the previous types of providers, the TraceLogging provider includes event format description into the recorded log data[8] allowing a consumer to easily understand the event data without prior knowledge of the format. As with manifest-based providers, TraceLogging can also be enabled by up to eight trace sessions simultaneously.

1.1.3 Consumers

Consumers are applications that consume events from providers. This is done through event *trace sessions*, where one session is created per provider. Consumers have the ability to both receive events in real time from *trace sessions*, or later on by events stored in log files. Furthermore, events can be filtered by many attributes such as timestamps.

1.1.4 Sessions

ETW sessions are created and managed by controllers to forwards events from one or more providers to a consumer such as the event log or simply a console output. As shown on Figure 1.1 (ETW model diagram[2]), sessions contain a number of buffers, one for each event provider. The session is responsible for these buffers, ie. the session creates and manages the buffer in its lifetime. Two predefined sessions exists in Windows, that is the *Global Logger Session* and the *NT Kernel Logger Session* handling events occurring early in the system boot process and predefined system events generated by the operating system respectively[5].

Figure 1.1 (ETW model diagram[2]) shows how the different components of ETW works together in sessions to produce and consume events.

1.1.5 Using Event Tracing for Windows (ETW)

As mentioned in **??** (**??**), the goal of this project is to research the possibilities of using built in telemetry, such as ETW, to detect the exploitation of vulnerabilities. Therefore, it is important to discover how ETW can be used to gather telemetry from providers.

One ETW provider that is widely used to detect malicious activity such as exploitation of vulnerabilities is the `Microsoft_windows-Threat-Intelligence`[3] provider. This is widely used by various Antivirus (AV) engines such as Microsoft's own Endpoint Detection and Response (EDR)/AV tool, Microsoft Defender for Endpoint. While this provider gives insight into Windows API calls often used in an exploitation process, we will not be focusing on this. As mentioned in **??** (**??**) and discussed in **??** (**??**), the project will revolve around detection of CVE-2020-24086, which is a vulnerability in the `tcpip.sys` driver of Windows. Due to this, we will in this section explore ETW providers relevant to this specific driver.

Finding providers

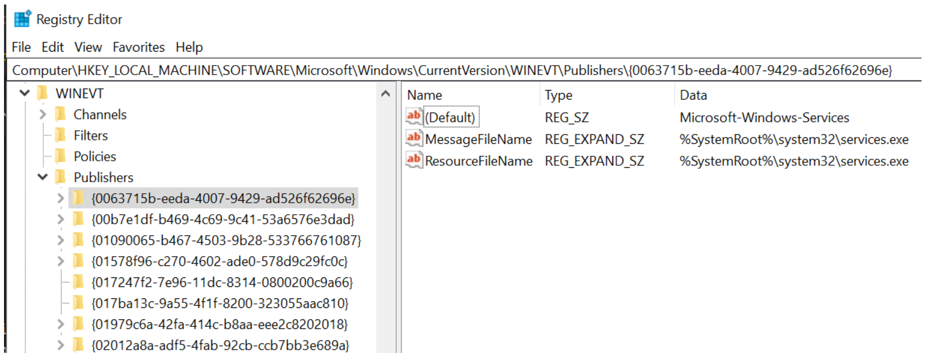
Read this section thorougly as it has been revised a lot ad-hoc

Getting a list of all available providers in Windows is fairly simply. A few methods exists, such as:

1. Using `logman query providers`

2. Using the PowerShell command `Get-TraceEtwProvider`
3. Enumerate registry keys under `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\WINEVT\Publishers`

The output from method *(3)*, the registry, is as mentioned in subsubsection 1.1.2 (Manifest-based providers), only for manifest-based providers. Therefore, not all providers will be shown here. Figure 1.2 (Finding ETW providers using Registry Editor) shows how the information available using Registry Editor. As it can be seen the registry contains information about the binary the provider is implemented in, which in our case is `tcpip.sys`.



Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\WINEVT\Publishers\{0063715b-eeda-4007-9429-ad526f62696e}

```
WINEVT
  Channels
  Filters
  Policies
  Publishers
    {0063715b-eeda-4007-9429-ad526f62696e}
    {00b7e1df-b469-4c69-9c41-53a6576e3dad}
    {01090065-b467-4503-9b28-533766761087}
    {01578f96-c270-4602-ade0-578d9c29fc0c}
    {017247f2-7e96-11dc-8314-0800200c9a66}
    {017ba13c-9a55-4f1f-8200-323055aac810}
    {01979c6a-42fa-414c-b8aa-eee2c8202018}
    {02012a8a-adf5-4fab-92cb-ccb7bb3e689a}
```

Name	Type	Data
(Default)	REG_SZ	Microsoft-Windows-Services
MessageFileName	REG_EXPAND_SZ	%SystemRoot%\system32\services.exe
ResourceFileName	REG_EXPAND_SZ	%SystemRoot%\system32\services.exe

Figure 1.2: Finding ETW providers using Registry Editor

To find all manifest-based providers we can use the PowerShell script on listing 2, where the output of the command is also shown.

```
Get-ChildItem -Path "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\WINEVT\Publishers"
    | Get-ItemProperty
    | Where-Object {$_.ResourceFileName -like '*tcpip.sys*'}
    | Format-List "(default)", ResourceFileName, MessageFilename, PSChildName

(default)        : Microsoft-Windows-TCPIP
ResourceFileName : C:\WINDOWS\system32\drivers\tcpip.sys
MessageFileName  : C:\WINDOWS\system32\drivers\tcpip.sys
PSChildName      : {2f07e2ee-15db-40f1-90ef-9d7ba282188a}
```

Listing 2: `logman query providers` output. See appendix ?? for full output

The same information queried using `logman query providers` can be seen in listing 3. However, the `logman query providers` command does not display the *ResourceFileName* as it can be seen on Figure 1.2 (Finding ETW providers using Registry Editor).

```
 1  Provider                                 GUID
 2  -------------------------------------------------------------------------------
 3  .NET Common Language Runtime             {E13C0D23-CCBC-4E12-931B-D9CC2EEE27E4}
 4  ACPI Driver Trace Provider               {DAB01D4D-2D48-477D-B1C3-DAAD0CE6F06B}
 5  Active Directory Domain Services: SAM    {8E598056-8993-11D2-819E-0000F875A064}
 6  Active Directory: Kerberos Client        {BBA3ADD2-C229-4CDB-AE2B-57EB6966B0C4}
 7  Active Directory: NetLogon               {F33959B4-DBEC-11D2-895B-00C04F79AB69}
 8  ADODB.1                                  {04C8A86F-3369-12F8-4769-24E484A9E725}
 9  ADOMD.1                                  {7EA56435-3F2F-3F63-A829-F0B35B5CAD41}
10  Application Popup                        {47BFA2B7-BD54-4FAC-B70B-29021084CA8F}
11  Application-Addon-Event-Provider         {A83FA99F-C356-4DED-9FD6-5A5EB8546D68}
12  ...
13  Microsoft-Windows-TCPIP                  {2F07E2EE-15DB-40F1-90EF-9D7BA282188A}
14  ...
15  TCPIP Service Trace                      {EB004A05-9B1A-11D4-9123-0050047759BC}
16  ...
```

Listing 3: `logman query providers` output. See appendix **??** for full output

The number of providers registered according to `logman query providers` is 1162 whereas 972 of these are manifest-based providers according to data from the registry. One example of a non-manifest-based provider that can only be found using logman is the provider *TCPIP Service Trace* as seen on line 15 in listing 3.

Using the second method, `Get-TraceEtwProvider`, simply yields a `Access Denied` error rendering it useless.

To conclude this section, we were able to find two different providers related to the TCP/IP stack on Windows. The first provider, *Microsoft-Windows-TCPIP* is definitely related to `tcpip.sys` according to the *ResourceFileName* property. The second however is a bit more cumbersome as logman does not provide any more information than the name (*TCPIP Service Trace*) and the GUID.

Starting a trace

In the previous sections we have explored the different components of ETW and how they work together. This section will showcase how to easily consume ETW events for specific providers. Once again, we are going to be using the *Microsoft-Windows-TCPIP* provider as an example, as this is the provider we will be exploring later on. To start a trace for *Microsoft-Windows-TCPIP* we need to take the following steps:

1. Start a new trace session that will hold our buffers
2. Add a provider to our trace session
3. Ensure that we can consume events produced in our trace, either through an event log or realtime

```
New-EtwTraceSession -Name TCPIPTrace -LogFileMode 0x8100
↪ -FlushTimer 1
Add-EtwTraceProvider -SessionName TCPIPTrace -Guid
↪ '{2F07E2EE-15DB-40F1-90EF-9D7BA282188A}' -MatchAnyKeyword 0x0
tracefmt -rt TCPIPTrace -displayonly
```

Listing 4: Starting a trace for *Microsoft-Windows-TCPIP - 2F07E2EE-15DB-40F1-90EF-9D7BA282188A*

In these three simple steps, we will be acting as the ETW controller through PowerShell to start a trace of *Microsoft-Windows-TCPIP*. Listing 4 shows how to do this using PowerShell and the `tracefmt` tool from Windows Driver Kit (WDK).

The code is explained here:

Explain bit-mask

New-EtwTraceSession Starts a new trace with the name *TCPIPTrace* and sets the `LogFileMode` to `EVENT_TRACE_USE_LOCAL_SEQUENCE` and `EVENT_TRACE_REAL_TIME_MODE`[6]. The `FlushTimer` argument states that the buffer should be flushed every second

Add-EtwTraceProvider Adds the *Microsoft-Windows-TCPIP* provider to the session matching any keywords (ie. we will consume all event keywords)

Maybe explain event keywords, and also how to get them

tracefmt Displays the content of the trace in the current console every time the buffer is flushed

At this point we are able to consume all events produced by `tcpip.sys`, and are therefore at a good position to consider how the data can be used for detection. However, in order to know exactly what to detect, it is important to first analyze CVE-2021-24086. In the next **??** (**??**) we will begin analyzing CVE-2021-24086 in order to understand exactly what we need to detect.

Rewrite this at some point

1.2 Function hooking

Maybe add other methods not used