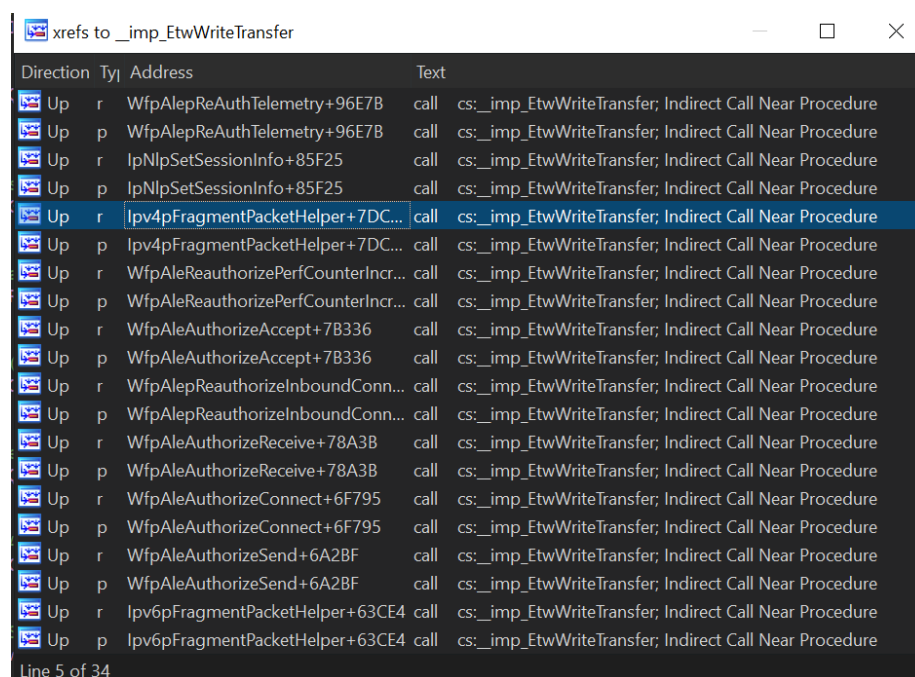


0.1 Event Tracing for Windows (ETW)

With Event Tracing for Windows (ETW), the provider has to provide events, meaning a call to one of the tracing APIs has to be there. When reverse engineering binaries it is possible to look for such events to discover if they are used. As we are mainly concerned about ETW events sent by `tcpip.sys` examining said binary is a good starting point. `tcpip.sys` registers itself as a Windows software trace preprocessor (WPP) provider and therefore uses the kernel mode functions `EtwWrite` and `EtwWriteTransfer`[3]. To examine the usage of `EtwWriteTransfer`, we can look at figure 1 where it shows that only 34 references to `EtwWriteTransfer` exists.



Direction	Ty	Address	Text
Up	r	WfpAlePReAuthTelemetry+96E7B	call cs:_imp_EtwWriteTransfer; Indirect Call Near Procedure
Up	p	WfpAlePReAuthTelemetry+96E7B	call cs:_imp_EtwWriteTransfer; Indirect Call Near Procedure
Up	r	IpNlpSetSessionInfo+85F25	call cs:_imp_EtwWriteTransfer; Indirect Call Near Procedure
Up	p	IpNlpSetSessionInfo+85F25	call cs:_imp_EtwWriteTransfer; Indirect Call Near Procedure
Up	r	Ipv4pFragmentPacketHelper+7DC...	call cs:_imp_EtwWriteTransfer; Indirect Call Near Procedure
Up	p	Ipv4pFragmentPacketHelper+7DC...	call cs:_imp_EtwWriteTransfer; Indirect Call Near Procedure
Up	r	WfpAleReauthorizePerfCounterIncr...	call cs:_imp_EtwWriteTransfer; Indirect Call Near Procedure
Up	p	WfpAleReauthorizePerfCounterIncr...	call cs:_imp_EtwWriteTransfer; Indirect Call Near Procedure
Up	r	WfpAleAuthorizeAccept+7B336	call cs:_imp_EtwWriteTransfer; Indirect Call Near Procedure
Up	p	WfpAleAuthorizeAccept+7B336	call cs:_imp_EtwWriteTransfer; Indirect Call Near Procedure
Up	r	WfpAlePReauthorizeInboundConn...	call cs:_imp_EtwWriteTransfer; Indirect Call Near Procedure
Up	p	WfpAlePReauthorizeInboundConn...	call cs:_imp_EtwWriteTransfer; Indirect Call Near Procedure
Up	r	WfpAleAuthorizeReceive+78A3B	call cs:_imp_EtwWriteTransfer; Indirect Call Near Procedure
Up	p	WfpAleAuthorizeReceive+78A3B	call cs:_imp_EtwWriteTransfer; Indirect Call Near Procedure
Up	r	WfpAleAuthorizeConnect+6F795	call cs:_imp_EtwWriteTransfer; Indirect Call Near Procedure
Up	p	WfpAleAuthorizeConnect+6F795	call cs:_imp_EtwWriteTransfer; Indirect Call Near Procedure
Up	r	WfpAleAuthorizeSend+6A2BF	call cs:_imp_EtwWriteTransfer; Indirect Call Near Procedure
Up	p	WfpAleAuthorizeSend+6A2BF	call cs:_imp_EtwWriteTransfer; Indirect Call Near Procedure
Up	r	Ipv6pFragmentPacketHelper+63CE4	call cs:_imp_EtwWriteTransfer; Indirect Call Near Procedure
Up	p	Ipv6pFragmentPacketHelper+63CE4	call cs:_imp_EtwWriteTransfer; Indirect Call Near Procedure

Line 5 of 34

Figure 1: References to `EtwWriteTransfer` in `tcpip.sys`

However, as `tcpip.sys` uses WPP, ETW is implemented through the use of macros[1] the real number cannot be found by direct references to `EtwWriteTransfer`. Instead we can create a graph of all functions interacting with `EtwWriteTransfer` somehow, which tells us that almost all functions has an indirect connection to `EtwWriteTransfer`. This gives a good overview of the scale of ETW usage within the `tcpip.sys` binary.

Hunting for relevant events. Looking at the stacktrace from listing ?? from ?? (??), we can see that we have a few function calls before the vulnerability is triggered in `Ipv6pReassembleDatagram`. Analyzing all ETW API calls we get the

results present on table 1, where we can see that while the top-level functions have four or five calls to ETW api functions, the lower level have none.

Function	# of ETW API calls	# due to error checking
Ipv6pReassembleDatagram	4	3
Ipv6pReceiveFragment	5	5
Ipv6pReceiveFragmentList	0	0
IppReceiveHeaderBatch	0	0
IppFlcReceivePacketsCore	0	0
IpFlcReceivePackets	0	0

Table 1: ETW API calls for functions related to CVE-2021-24086

Event usability analysis. While we now know that ETW events are created in certain circumstances within the relevant functions of `tcpip.sys`, the question remains whether any of these events can be used for detecting attempts to exploit CVE-2021-24086. If we look closer at table 1 we can see that most of the ETW api calls are a result of error checking. An example of such a check can be seen on listing 1. As it clearly shows, a event is written if the allocation of a `NetBufferList` fails. The other error checking events are very similar in nature. This poses an issue in regards to detection, as exploits usually exploit *unknown* states in binaries that can often be abused for malicious purposes. As the states exploited are unknown, default error checking do usually not detect such states, because if they did there would most likely be no vulnerability. This is true for the error check shown in listing 1, and it is also true for the eight other events written in `Ipv6pReassembleDatagram` and `Ipv6pReceiveFragment`.

0.1. Event Tracing for Windows (ETW)

```
1 NetBufferList = (_NET_BUFFER_LIST *)NetioAllocateAndReferenceNetBufferA_
  ↳ ndNetBufferList(IppReassemblyNetBufferListsComplete, datagram,
  ↳ 0i64, 0i64, 0, 0);
2 if ( !NetBufferList )
3 {
4     LOBYTE(v12) = a3;
5     IppDeleteFromReassemblySet(v7 + 20304, datagram, v12);
6     goto failure;
7 }
8
9 ...
10
11 failure:
12     if ( dword_1C01FECA4 == 1 &&
  ↳ (WPP_MAIN_CB.Queue.Wcb.NumberOfChannels & 0x8000000) != 0 )
13         McTemplateK0z_EtwWriteTransfer(
14             (__int64)&MICROSOFT_TCPIP_PROVIDER_Context,
15             (__int64)&TCPIP_MEMORY_FAILURES,
16             v14,
17             L"IPv6 reassembly structures (IPNG)");
18
19     goto memory_failure;
```

Listing 1: Example of error checking ETW api call

The last event which is not directly related to error checking, at least not on the surface, is unfortunately an error checking event in disguise. The event is shown on listing 2. Knowing the specification of `EtwWriteTransfer`[2], we can quickly see that the second parameter, `PCEVENT_DESCRIPTOR EventDescriptor`, shows that this event is also most likely related to some kind of error[2]. The value is `TCPIP_IP_REASSEMBLY_FAILURE_IPSEC`[4] strongly suggesting that it is only triggered upon a failure. Looking at the event known to be error checking on listing 1 we can also see the similarity between the naming of the `EventDescriptor`.

```
1 if ( (BYTE4(WPP_MAIN_CB.Queue.Wcb.DeviceRoutine) & 0x40) != 0 &&
  ↳ dword_1C01FECA4 == 1 )
2     McTemplateK0qqq_EtwWriteTransfer(
3         (unsigned int)&MICROSOFT_TCPIP_PROVIDER_Context,
4         (unsigned int)&TCPIP_IP_REASSEMBLY_FAILURE_IPSEC,
5         (unsigned int)&MICROSOFT_TCPIP_PROVIDER,
6         *(_DWORD *) (v6 + 8),
7         23,
8         v20);
```

Listing 2: Event presumably unrelated to error checking

0.1. Event Tracing for Windows (ETW)

Examining the surrounding code further, we can see that the event is sent if the `NetBuffer` used to store the packet is not large enough to fit the packet, thereby concluding that is actually is an error checking event.

Having examined all the events present in any of the related functions from the stacktrace we can with confidence say that it is not possible to detect CVE-2021-24068 using close proximity ETW events.

0.1.1 Usage of ETW for known vulnerabilities

In the previous section we highlighted the usage of ETW events used to report when the execution enters into unintended states, and how this cannot be used for detecting unknown vulnerabilities. A change that we did not highlight during the analysis of CVE-2021-24086 in ?? (??), was the fact that a new call to `EtwWriteTransfer` was added in the patched version of `tcpip.sys`. The call can be seen in listing 3 and is written when the size of the packet is above 0xFFFF bytes.

Maybe add paragraph about why looking at other ETW events is the same as network analysis which is probably easier

```
1  if ( packetSize > 0xFFFF )
2  {
3      if ( (BYTE4(WPP_MAIN_CB.Queue.Wcb.DeviceRoutine) & 0x40) != 0 &&
4          ↪ dword_1C01FECA4 == 1 )
5          McTemplateK0qq_EtwWriteTransfer(
6              &MICROSOFT_TCPIP_PROVIDER_Context,
7              &TCPIP_IP_REASSEMBLY_FAILURE_PKT_LEN,
8              &MICROSOFT_TCPIP_PROVIDER,
9              *(unsigned int *)(v6 + 8),
10              23);
11
12  LOBYTE(LockArray_high) = a3;
13  IppDeleteFromReassemblySet(v7 + 20304, a2, LockArray_high);
14  goto failure;
15 }
```

Listing 3: ETW event in patched version of `Ipv6pReassembleDatagram`

Detecting CVE-2021-24086 attempts post-patch. To showcase the power of ETW we can examine this specific event and create a simple ETW session to consume it.

To understand how to filter ETW events it is important to understand the `EVENT_DESCRIPTOR` structure as seen on listing 4, notably the parameters `UCHAR Level` and `ULONGLONG Keyword`¹

¹ULONGLONG is equivalent to a 64-bit unsigned integer, `uint64`

0.1. Event Tracing for Windows (ETW)

```
1 typedef struct _EVENT_DESCRIPTOR {
2     USHORT    Id;
3     UCHAR     Version;
4     UCHAR     Channel;
5     UCHAR     Level;
6     UCHAR     Opcode;
7     USHORT    Task;
8     ULONGLONG Keyword;
9 } EVENT_DESCRIPTOR, *PEVENT_DESCRIPTOR;
```

Listing 4: EVENT_DESCRIPTOR structure[4]

Both the `Level` and `Keyword` parameters are used to filter events when using the PowerShell function `Add-EtwTraceProvider`. Looking at the specific event written, we can see that the `Level` is equal to *Warning*[4] and the `Keyword` is equal to *0x8000000200000080*. Listing 5 shows exactly how to start a trace for this specific event and Listing 6 shows the result when trying to exploit CVE-2021-24086 on a patched Windows 10 machine.

```
1 New-EtwTraceSession -Name TCPIPTrace -LogFileName 0x8100 -FlushTimer 1
2 Add-EtwTraceProvider -SessionName TCPIPTrace -Guid
   ↳ '{2F07E2EE-15DB-40F1-90EF-9D7BA282188A}' -MatchAnyKeyword
   ↳ "0x8000000200000080" -Level 0x2
3 .\tracefmt -rt TCPIPTrace -displayonly
```

Listing 5: Starting an ETW session to detect CVE-2021-24086 post-patch

```
1 [2]0000.0000::07/08/2021-22:37:02.308
   ↳ [Microsoft-Windows-TCPIP]Reassembly failure: packets do not add up
   ↳ correctly. Interface = 6. Address family = IPV6 .
```

Listing 6: Post-patch detection of CVE-2021-24086 exploitation attempt

As it can be seen, the message *"Reassembly failure: packets do not add up correctly"* is displayed which is an event generated by `void`.

Write about how we can only detect post-patch, but that this also proves that ETW can be used, if only the "right" events are written