# 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

June 1, 2021

# Todo list

**Abstract**

Write something very clever here and read it through 10000 times

# Table of contents

# Introduction

Introduce something here

## 1.1 Purpose

Purpose

## 1.2 Thesis overview

Thesis overview

## 1.3 Related work

Purpose

# Tracing and logging

## 2.1 Windows telemetry

## 2.2 Event tracing for Windows

In the architecture of Event Tracing for Windows (ETW) events are at the centerpiece where they are created, managed and consumed by different event components[4]. These differentiate between event *providers*, event *consumers*, and event *controllers*. All of these event components handle the workflow of ETW, either by reading or writing, or by controlling the events in some way.

Add proper description of what ETW is.

Mention that we do not look at this from a developers perspective, but from analyzing already made applications/drivers

figure out if the components should just be subsections

### 2.2.1 Event components

#### Controllers

#### 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.

Microsoft define four different types of providers depending on the version of Windows and type of application you are interested in.

#### Managed Object Format (MOF) (classic) providers

#### Windows softwarre trace preprocessor (WPP) providers

#### Manifest-based providers

#### TraceLogging providers

#### Consumers

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

Figure shows how the different components of ETW works together to produce and consume events

add ref to figure

### 2.2.2 Finding providers

### 2.2.3 Consuming events

# Vulnerability analysis

## 3.1 CVE-2021-24086

According to Microsoft[5] 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 [7] 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)[6] security software providers are given early access to vulnerability information. This information often include Proof of Concept (PoC)s for vulnerabilities to be patched, in order to aid security software providers to create 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[10] and Palo Alto[9] posted public information about CVE-2021-24086. However, both articles contained very limited details, and is therefore far from sufficient to reproduce the vulnerability. Before trying to rediscover the vulnerability, the following information is available:

- The vulnerability lies within the handling om fragmented packets in IPv6
- The relevant code lies within the `tcpip.sys` drivers
- 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[8][11], and has been made popular and easy to do by tools such as Bindiff[12] and Diaphora[3].

write a little about how bindiffing works. Or don't idc.

If we look at figure 3.1 we can compare the function changes of the patched and not-patched `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 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 package 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 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 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 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 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 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

**IPv6 Extension headers**

[2, sec. 4.5]

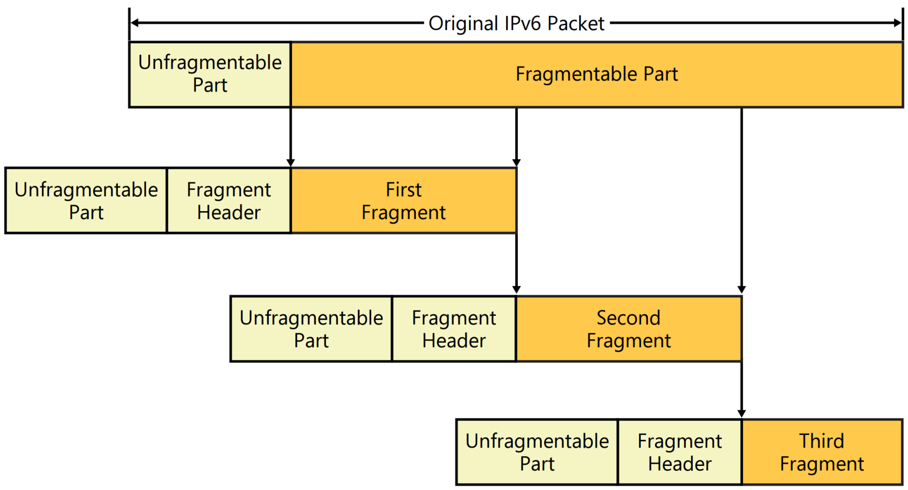


Figure 3.2: IPv6 fragmentation[1]

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

| | | | | |
|---|---|---|---|---|
| Next header | Reserverd | Fragment offset | Res | M |
| Identification | | | | |

$\Big\}$ Fragment header

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

**QUESTION_NAME** is the compressed name of the NetBIOS name for the request.

Figure 3.3: IPv6 Fragment Header [2, sec. 4.5]

#### IPv6 Fragment header

### 3.1.4 Root-cause analysis

### 3.1.5 Triggering the vulnerability

# Detection

## 4.1 Event Tracing for Windows (ETW)

## 4.2 Hooking and DTrace

## 4.3 Implementation

# Scaling and extensibility

# Conclusion

Conclude something please

# Abbreviations

**ETW** Event Tracing for Windows. 3, 4, 11

**MAPP** Microsoft Active Protetions Program. 5

**MOF** Managed Object Format. 3

**PoC** Proof of Concept. 5

**WPP** Windows softwarre trace preprocessor. 3

# Bibliography


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

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

[3] 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).

[4] 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).

[5] 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).

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

[7] 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).

[8] 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).

[9] 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).

[10] 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).

[11] 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).

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

# List of Figures

# List of code snippets

# Appendices

Fix appendices title location