

## 0.1 CVE-2021-24086

According to Microsoft[4] 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 [6] 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

### 0.1.1 Public information

Due to the Microsoft Active Protections Program (MAPP)[5] 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[9] and Palo Alto[8] 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 of 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

### 0.1.2 Binary diffing

The usage of binary diffing to gather information about patched vulnerabilities is well described in current research[7][10], and has been made popular and easy to do by tools such as Bindiff[11] and Diaphora[3].

If we look at figure 1 we can compare the function changes of the patched and not-patched `tcpip.sys`. Looking at `tcpip!Ipv6ReassembleDatagram` we can see that the similarity factor is only 0.38 telling us that a significant amount of code has been changed.

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

## 0.1. CVE-2021-24086

---

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	lppCleanupPathPrimitive
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 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 0.1.1 (Public information), that the vulnerability is triggered by a package of around `0xFFFF` bytes.

```
1  --- "a/.\\unpatched tcpip.sys"
2  +++ "b/.\\patched tcpip.sys"
3  @@ -1,6 +1,4 @@
4  -sub     rsp, 58h          ; Integer Subtraction
5  +sub     rsp, 60h          ; Integer Subtraction
6  movzx   r9d, word ptr [rdx+88h] ; Move with Zero-Extend
7  mov     rdi, rdx
8  mov     edx, [rdx+8Ch]
9  -mov     bl, r8b
10 +mov     r13b, r8b
11 add     edx, r9d          ; Add
12 -mov     byte ptr [rsp+98h+var_70], 0
13 -and     [rsp+98h+var_78], 0 ; Logical AND
14 mov     [rsp+98h+length], edx
15 lea     eax, [rdx+28h]    ; Load Effective Address
16 -mov     rdx, rdi
17 mov     [rsp+98h+var_68], eax
18 lea     eax, [r9+28h]     ; Load Effective Address
19 mov     [rsp+98h+BytesNeeded], eax
20 -xor     r9d, r9d         ; Logical Exclusive OR
21 mov     rax, [rcx+0D0h]
22 -lea     rcx, IppReassemblyNetBufferListsComplete ; Load
    ↪ Effective Address
23 -mov     r13, [rax+8]
24 -mov     rax, [r13+0]
25 +mov     r12, [rax+8]
26 +mov     rax, [r12]
27 mov     r15, [rax+28h]
28 mov     eax, gs:1A4h
29 mov     r8d, eax
30 -mov     rax, [r13+388h]
31 +mov     rax, [r12+388h]
32 lea     rbp, [r8+r8*2]    ; Load Effective Address
33 -mov     r12, [rax+r8*8]
34 -xor     r8d, r8d         ; Logical Exclusive OR
35 +mov     rcx, [rax+r8*8]
36 shl     rbp, 6           ; Shift Logical Left
37 -add     rbp, [r15+4728h] ; Add
38 +add     rbp, [r15+4728h] ; Add
39 +mov     [rsp+98h+var_58], rcx
40 +cmp     edx, 0FFFFFFh    ; Compare Two Operands
41 +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,
4  ↪ struct_datagram *datagram, char a3) {
5  unfragmentableHeaderLength =
6  ↪ datagram->unfragmentableHeaderLength;
7  packetSize = unfragmentableHeaderLength +
8  ↪ datagram->fragmentableLength;
9  BytesNeeded = unfragmentableHeaderLength + 40;
10 v6 = *(_QWORD *)(*(_QWORD *) (a1 + 208) + 8i64);
11 v7 = *(_QWORD *)(*(_QWORD *) v6 + 40i64);
12 LockArray_high = HIDWORD(KeGetPcr()[1].LockArray);
13 -v11 = NetioAllocateAndReferenceNetBufferAndNetBufferList(IppRea_
14 ↪ ssemblyNetBufferListsComplete, datagram, 0i64, 0i64, 0,
15 ↪ 0);
16 +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.

### 0.1.3 IPv6 fragmentation primer

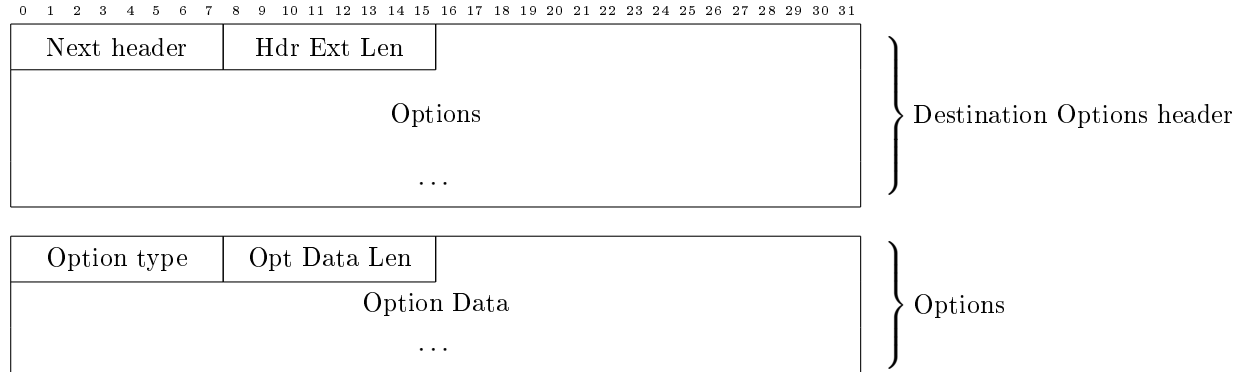
When the size of a packet is larger than the Maximum transmission unit (MTU) of the outbound interface, IPv6 fragmentation is used. The MTU of most standard network equipment and desktop computers is 1500 bytes. Therefore if you have an IPv6 packet that is larger than 1500 bytes, the packet must be fragmented. 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[2, 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 0.1.1, 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 would be the device that we are trying to attack using CVE-2021-24086. The specification can be seen on Figure 2 (IPv6 Destination Options Header [2, 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 2: IPv6 Destination Options Header [2, sec. 4.6]

By default, only one option exists, the *PadN option*[2, 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

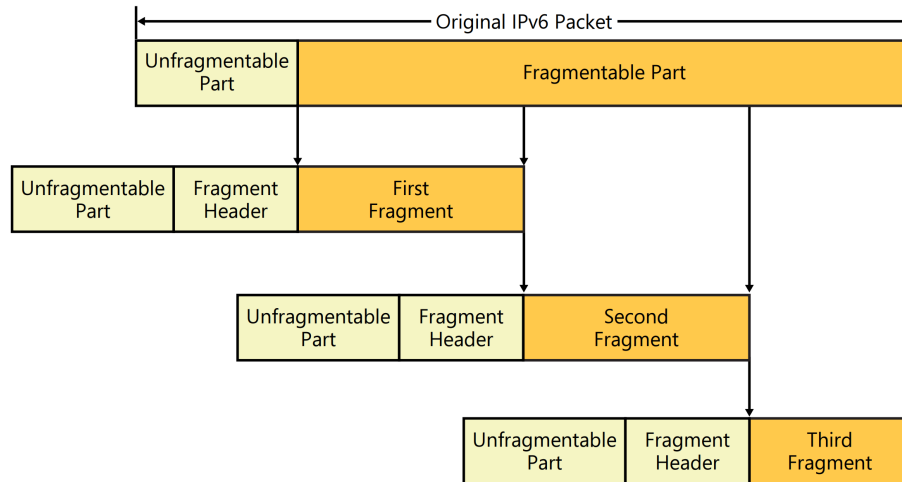
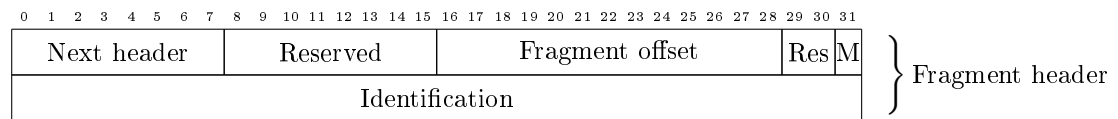


Figure 3: IPv6 fragmentation[1]



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.

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

Figure 4: IPv6 Fragment Header [2, sec. 4.5]

**IPv6 Fragment header**

**0.1.4 Root-cause analysis**

**0.1.5 Triggering the vulnerability**