0.1 Function hooking

Detecting CVE-2021-24086 using function hooking. As explained in ?? (??) 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 redirection 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[2].
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 1, 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`.

```
1  movzx   r9d, word ptr [rdx+88h] ; Move with Zero-Extend
2  mov     edx, [rdx+8Ch]
3  mov     r13b, r8b
4  add     edx, r9d        ; Add
5  ...
6  cmp     edx, 0FFFFh     ; Compare Two Operands
```

Listing 1: 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. Translating this into a C-style function suitable as the target of a function hook will give the result as seen on listing 2

```
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 2: `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 **??** (**??**), we might even be able to hijack the execution effectively preventing the exploit from stealing the execution.

0.1.1 Implementation

Implementing a full-fledged kernel function hooking driver requires substantial amount of time and effort. As discussed in **??** (**??**), 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 against. As the goal of this project is to investigate whether or not it is possible to detect vulnerabilities using information gained from patches, we deem it unnecessary to implement a full kernel driver to do prove so. Instead of developing a kernel driver, we decided to implement a WinDbg expression[1] 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 3.

```
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 2

.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 Returns the execution if the condition is not met

Listing 3: 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 led 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

4

```
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}"
breakpoint 0 redefined
4: kd> g
Packet size:
Evaluate expression: 63720 = 00000000`0000f8e8
Packet size:
Evaluate expression: 63731 = 00000000`0000f8f3
Packet size:
Evaluate expression: 65528 = 00000000`0000fff8
Packet size:
Evaluate expression: 65539 = 00000000`00010003
CVE-2021-24086 exploitation attempt detected

KDTARGET: Refreshing KD connection

*** Fatal System Error: 0x000000d1
                       (0x0000000000000000,0x0000000000000002,0x0000000
                       ↪  000000001,0xFFFFF8060D2A937B)

Break instruction exception - code 80000003 (first chance)

A fatal system error has occurred.
Debugger entered on first try; Bugcheck callbacks have not been invoked.

A fatal system error has occurred.

For analysis of this file, run !analyze -v
nt!DbgBreakPointWithStatus:
fffff806`091fc440 cc              int     3
```

Listing 4: Detection of CVE-2021-24086 using an WinDbg expression